Industrial Training Report on

Flexgrid in WPF

A Thesis Submitted

In Partial Fulfilment of the Requirement

For the Degree of

MASTER OF COMPUTER APPLICATION

Submitted by

Nitin Sharma – (1900290149069)

Under the Supervision of

Prof. Ankit Verma

(Assistant Professor)

Department of Computer Applications,

KIET Group of Institutions,

Delhi-NCR, Ghaziabad

to the

Faculty of Computer Application

DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY

LUCKNOW

(Formerly Uttar Pradesh Technical University, Lucknow)June

2021

DECLARATION

        I hereby declare that the work presented in this report entitled "Flexgrid in WPF", was carried out by me. I have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute.

I have given due credit to the original authors/sources for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. I have used quotation marks to identify verbatim sentences

and given credit to the original authors/sources.

I affirm that no portion of my work is plagiarized, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, I shall be fully

responsible and answerable.

Name : Nitin Sharma

Roll. No. : 1900290149069

Course/Branch : MCA

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

FlexGrid for WPF is a lightweight data grid control designed on a flexible object model. Based on the popular WinForms version, **FlexGrid** offers many unique features such as unbound mode, flexible cell merging, and multi-cell row and column headers that make it a proven solution for data management and tabulation.

Like other data visualization controls, FlexGrid supports bound mode and can be populated with data from various data sources. The below section discusses bound FlexGrid in .NET 4.5.2 and .NET 5 versions.

The C1FlexGrid class provides the **ItemsSource** property to populate data in the grid. In WPF, the ItemsSource property binds FlexGrid to **IEnumerable** interface, or to DataCollection interface.
The following image shows a bound FlexGrid.

The following code examples illustrate how to bind a list of customer objects to FlexGrid control. The code below causes the grid to scan the data source and automatically generate columns for each public property of the items in the data source. You can also customize automatically created columns using code, or disable automatic column generation altogether and create custom columns through code or XAML.

FlexGrid can also be bound directly to a list (customer list). However, binding to **DataCollection** is usually better as it retains a lot of the data configuration for the application, which can be shared across controls.
The FlexGrid control is designed to work with various data sources and collections such as ObservableCollection and DataCollection to leverage its full capabilities. However, the control is not restricted to data sources and can be used in unbound mode To create an unbound grid, add rows and columns to the grid using the Add method. The following code illustrates adding rows and columns in a grid and populating it with an indexing notation that specifies a cell by corresponding row and column index.

FlexGrid offers several advanced data visualization features that are beyond simple grids. These features are listed below:

- **Flexible   data   binding**
  FlexGrid can be used in bound mode, where it displays data from a data source, or in unbound mode, where the grid itself manages the data.
- **Advanced   grid   features**
  FlexGrid supports advanced grid features including cell merging, data filtering, sorting, editing, aggregation etc. You can merge contiguous, like-valued cells to make the data span across multiple cell; calculate totals averages, and other statistics for ranges of cells; and apply filters to each column on the grid.
- **Hierarchical   styles**
  FlexGrid summarizes data in hierarchical style like a tree. Each record can be expanded or collapsed to expose details in child grids.
- **Integrated   printing   support**
  FlexGrid supports integrated printing wherein it has control over paper orientation, margins, and footer text. With a rich object model, the control provides varied printing events to handle page breaks, add repeating header rows, or add custom elements to each page. You can also show a dialog to let users select and set up the printer.
- **Advanced   grouping   and   filtering   feature**
  FlexGrid supports grouping as a UI feature through a separate control, FlexGridGroupPanel, available as a separate assembly. Similarly, the control comes with FlexGridFilter component, which provides ad-hoc filtering and is shipped separately to achieve lower footprint.
- **Custom   cells**
  FlexGrid supports significant customization in the grid through custom cells. The control provides CellFactory class and built-in CellTemplate and CellEditingTemplate to customize visual elements.
- **Row   details**
  FlexGrid provides the flexibility to show row details in a data template, which can be used to display text, images as well as data bound controls.
- **Freezing   and   Pinning**
  This feature allows you to freeze the rows and columns using mouse drag during runtime. A column or multiple columns can be pinned to the left-hand side of the FlexGrid. Column Pinning in FlexGrid allows the you to lock column in a particular column order, this will allow you to see it while horizontally scrolling the Grid.

CHAPTER 2

TECHNOLOGIES / SOFTWARE REQUIREMENTS

- Operating System : Windows

- Programming language: C#, WPF(.Net Framework)

- Front-End: XAML

- Back-End: .Net Framework DataTable and JSON

- IDE: Visual Studio 2019

- API Control: Component One Flexgrid

CHAPTER 3

HARDWARE REQUIREMENT / HARDWARE USED

- Windows Version- Windows.

- 2 GB Ram

- 1MB Cache Memory

- External Memory 10 GB

CHAPTER 4

FEATURING AND IMAGES

FlexGrid for WPF is a lightweight data grid control designed on a flexible object model. Based on the popular WinForms version, **FlexGrid** offers many unique features such as unbound mode, flexible cell merging, and multi-cell row and column headers that make it a proven solution for data management and tabulation.

API References	
C1.WPF.FlexGrid.4.5.2 Assembly	C1.WPF.Grid Assembly

Note: ComponentOne FlexGrid control is compatible with both **.NET 4.5.2** and **.NET 5 Frameworks.**

Like other data visualization controls, FlexGrid supports bound mode and can be populated with data from various data sources. The below section discusses bound FlexGrid in .NET 4.5.2 and .NET 5 versions.

The C1FlexGrid class provides the **ItemsSource** property to populate data in the grid. In WPF, the ItemsSource property binds FlexGrid to **IEnumerable** interface, or to DataCollection interface.
The following image shows a bound FlexGrid.

ID	Name	Country	Country ID	First	Last
0	Ben Evers	India	1	Ben	Evers
1	Ed Saltzman	Myanmar	4	Ed	Saltzman
2	Gil Rodriguez	Japan	3	Gil	Rodriguez
3	Fred Rodriguez	United States	2	Fred	Rodriguez
4	Dan Ambers	Japan	3	Dan	Ambers
5	Elena Evers	China	0	Elena	Evers
6	Charlie Rodriguez	China	0	Charlie	Rodriguez
7	Herb Spencer	India	1	Herb	Spencer
8	Alaric Evers	China	0	Alaric	Evers
9	Andy Evers	China	0	Andy	Evers
10	Alaric Spencer	Japan	3	Alaric	Spencer
11	Elena Ambers	Myanmar	4	Elena	Ambers

The following code examples illustrate how to bind a list of customer objects to FlexGrid control. The code below causes the grid to scan the data source and automatically generate columns for each public property of the items in the data source. You can also customize automatically created columns using code, or disable automatic column generation altogether and create custom columns through code or XAML.

```
grid.ItemsSource = Customer.GetCustomerList(12);
```

FlexGrid can also be bound directly to a list (customer list). However, binding to **DataCollection** is usually better as it retains a lot of the data configuration for the application, which can be shared across controls.

The FlexGrid control is designed to work with various data sources and collections such as ObservableCollection and DataCollection to leverage its full capabilities. However, the control is not restricted to data sources and can be used in unbound mode.

The image given below shows an unbound grid populated with cell index notation.

[0,0]	[0,1]	[0,2]	[0,3]	[0,4]
[1,0]	[1,1]	[1,2]	[1,3]	[1,4]
[2,0]	[2,1]	[2,2]	[2,3]	[2,4]
[3,0]	[3,1]	[3,2]	[3,3]	[3,4]
[4,0]	[4,1]	[4,2]	[4,3]	[4,4]
[5,0]	[5,1]	[5,2]	[5,3]	[5,4]
[6,0]	[6,1]	[6,2]	[6,3]	[6,4]
[7,0]	[7,1]	[7,2]	[7,3]	[7,4]
[8,0]	[8,1]	[8,2]	[8,3]	[8,4]
[9,0]	[9,1]	[9,2]	[9,3]	[9,4]
[10,0]	[10,1]	[10,2]	[10,3]	[10,4]

To create an unbound grid, add rows and columns to the grid using the Add method. The following code illustrates adding rows and columns in a grid and populating it with an indexing notation that specifies a cell by corresponding row and column index.

The indexing notation displayed in the grid specifies a cell by row and column index. The cells can also be specified by row index and column name, or by row index and column name. The indexing notation works in bound and unbound modes. In bound mode, the data is retrieved or applied to the items in the data source. In unbound mode, the data is stored internally by the grid.

The new indexing notation displayed in the grid contains no items in the 0th row. This notation makes indexing easier as the indices match the index of data items and the column count matches the number of displayed properties. The only drawback of this notation is that a new method is required to access the content of fixed cells

Most grid controls allow users to select parts of the data using the mouse and the keyboard. FlexGrid supports following selection modes through the **SelectionMode** property of C1FlexGrid class in .NET 4.5.2 version and GridBase class in .NET 5 version:

- **Cell**: To select a single cell.
- **CellRange**: To select a cell range (block of adjacent cells).
- **Row**: To select an entire row.
- **RowRange**: To select a set of contiguous rows.

- **ListBox**: To select an arbitrary set of rows (not necessarily contiguous).

The default **SelectionMode** is **CellRange**, which provides an Excel-like selection behavior. The row-based options are also useful in scenarios where it makes sense to select whole data items instead of individual cells. Regardless of the selection mode, FlexGrid exposes the current selection with the Selection property. This property gets or sets the current selection as a **CellRange** object.

Active	Name	Country	Country ID	First Name
☐	Gil Saltzman	China	0	Gil
☐	Charlie Cole	India	1	Charlie
☐	Gina Evers	China	0	Gina
☐	Jim Evers	United States	2	Jim
☐	Fred Salvatore	China	0	Fred
☐	Andy Ambers	China	0	Andy
☐	Andy Spencer	United States	2	Andy
☐	Fred Evers	India	1	Fred

FlexGrid includes two features that allow you to customize the way in which the selection is highlighted for the user.

Excel-Style Marquee

If you set the ShowMarquee property to true, the grid automatically draws a rectangle around the selection, making it extremely easy to see. By default, the marquee is a two-pixel thick black rectangle, but you can customize it using the Marquee property.
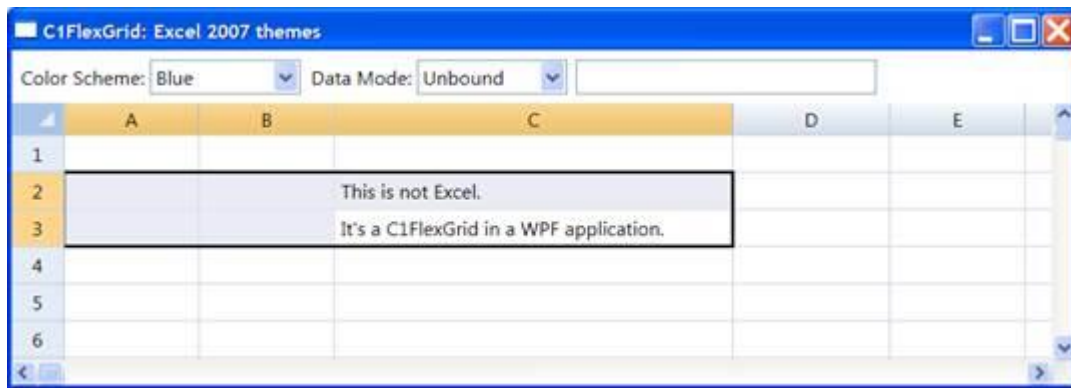
FlexGrid designer provides a context menu with options to select Excel-like color schemes (Blue, Silver, Black). In addition, you can easily copy the XAML generated by the designer into reusable style resources.

FlexGrid control by default lets you to **sort** by any column. The user simply has to click on the header of a column to toggle between the sorting states, ascending and descending.

If a user clicks the column header, it sorts by that column. In this case, the column header shows an upward pointing arrow indicating ascending order. Likewise, if a user clicks the column header once more, that is, the second time, the column sorts in descending order, with the arrow pointing downwards.

The snapshot below depicts sorting feature by the "Description" and "UnitPrice" columns:

FlexGrid supports cell merging to allow data to span across multiple rows and columns. The cell merging capability can be used to enhance the appearance of data.

In the following section learn how to implement cell merging in FlexGrid .NET 4.5.2 and .NET 5 versions.

Cell merging can be enabled by setting the **AllowMerging** property of C1FlexGrid in code. To merge columns, you need to set the AllowMerging property for each column that you want to merge to **true**.

Similarly, to merge rows, set the AllowMerging property to **true** for each row to be merged. You can use **AllowMerging Enum** to specify areas like **Cells**, **ColumnHeaders** etc. of the grid for merging.

The following image shows merged columns in a FlexGrid.

FlexGrid displays various icons during its operations such as sorting, filtering etc. These icons can be changed using various icon templates provided in the FlexGrid control. These icon templates can be accessed through following properties.

You can change the icons set by these templates either to the built-in icons provided by the FlexGrid or to your own custom image, geometric figures, font etc as an icon.

The following image displays a custom image which is set as a sort icon for sorting values in descending order.

ID	Name	CountryID	Active	First	Last
24	Zeb Stevens	Italy	☑	Zeb	Stevens
86	Zeb Quaid	Mexico	☑	Zeb	Quaid
37	Zeb Lehman	Italy	☐	Zeb	Lehman
39	Zeb Frommer	Italy	☐	Zeb	Frommer
71	Zeb Danson	Turkey	☑	Zeb	Danson
10	Xavier Neiman	Mexico	☐	Xavier	Neiman
60	Xavier Lehman	France	☐	Xavier	Lehman
38	Xavier Krause	Egypt	☑	Xavier	Krause
49	Xavier Frommer	Pakistan	☐	Xavier	Frommer
78	Xavier Frommer	Philippines	☑	Xavier	Frommer
96	Xavier Cole	Italy	☑	Xavier	Cole
9	Vic Stevens	Brazil	☐	Vic	Stevens
19	Vic Richards	Pakistan	☐	Vic	Richards
20	Vic Griswold	Mexico	☑	Vic	Griswold
23	Vic Bishop	Ethiopia	☐	Vic	Bishop
36	Ulrich Richards	United States	☑	Ulrich	Richards
28	Ulrich Griswold	United Kingdom	☑	Ulrich	Griswold
5	Ulrich Evers	Mexico	☐	Ulrich	Evers
14	Ulrich Cole	Thailand	☐	Ulrich	Cole
48	Ted Stevens	Iran	☑	Ted	Stevens
53	Ted Richards	Nigeria	☐	Ted	Richards
30	Ted Neiman	Japan	☑	Ted	Neiman
4	Ted Myers	Nigeria	☐	Ted	Myers

FlexGrid also allows you to change the appearance of the different icons used in the control using the C1Icon class. The C1Icon class is an abstract class that provides a series of different objects that can be used for displaying monochromatic icons which can easily be tinted and resized.

FlexGrid supports grouping through I**DataCollection** interface. You can create hierarchical views in FlexGrid by defining each level of grouping through the **PropertyGroupDescription** class. Using the PropertyGroupDescription object, you can select the property to group data, and implement **ValueConverter** to determine how to use property value while grouping. You can also disable grouping at the grid level by setting the grid's GroupRowPosition property to **None**.

The following image shows data grouped by country and their active state. Users can click the icons on group headers to collapse or expand the groups, as they would do with a TreeView control.

Country	Active	ID	Name	Country ID	First Name	Last Name
⌄ Country: China (13 items)						
⌄ Active: False (13 items)						
China	☐	0	Alaric Rodriguez	0	Alaric	Rodriguez
China	☐	4	Gina Frommer	0	Gina	Frommer
China	☐	5	Ed Salvatore	0	Ed	Salvatore
China	☐	14	Herb Rodriguez	0	Herb	Rodriguez
China	☐	18	Ed Bishop	0	Ed	Bishop
China	☐	20	Ben Saltzman	0	Ben	Saltzman
China	☐	25	Fred Frommer	0	Fred	Frommer
China	☐	26	Fred Frommer	0	Fred	Frommer
China	☐	27	Herb Ambers	0	Herb	Ambers
China	☐	31	Charlie Evers	0	Charlie	Evers
China	☐	32	Herb Danson	0	Herb	Danson
China	☐	37	Alaric Bishop	0	Alaric	Bishop

The following code example illustrates data grouping by country and active state through DataCollection.

FlexGrid provides Excel-like filtering feature to filter data through drop down icons in column headers. This functionality is available in FlexGrid through a separate control called FlexGridFilter, which is implemented through an extender assembly, **C1.WPF.FlexGridFilter**. Once the FlexGridFilter control is added, the grid displays a drop-down icon on hovering the column headers. The drop-down icon shows an editor that allows users to specify filters on columns. Users may choose between the two types of filters:

- **Value filter**: This filter lets you filter specific values in the column.
- **Condition filter**: This filter lets you specify conditions composed of an operator (greater than, less than, etc.) and a parameter. The conditions can be combined using an **AND** or an **OR** operator.

-

Note: Filtering using **FlexGridFilter** is only availabe in .NET 4.5.2 version.

The images below show the filters displayed on clicking the drop-down icon.
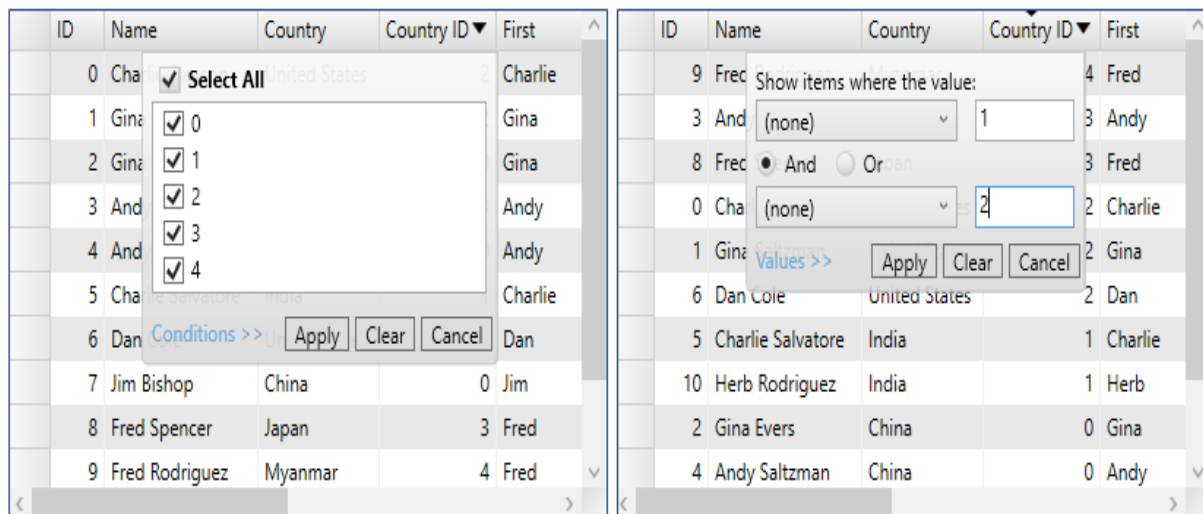
Value Filter

Conditional Filter

FlexGrid lets you compute and display aggregate value for each group created after grouping data. The control shows groups in a collapsible outline format and automatically displays the number of items in each group. However, you can go one step further and display aggregate values for every grouped column. This feature enables users in drawing out more insights from random data.

In the following section learn how to implement Data Aggregation in FlexGrid .NET 4.5.2 and .NET 5 versions.

In the below example, a company's sales data grouped by country or product category can be more useful if the aggregate sales can be indicated against each country and product category in the grid itself. The Column class provides the **GroupAggregate** property that can be set to **Aggregate** to automatically calculate and display aggregates. The aggregates calculated by setting the GroupAggregate property automatically recalculate the aggregate values when the data changes.

The following image shows a FlexGrid with aggregate values displayed in the columns.

Line	Color	Name	Price	Cost	Weight	Volume
▲ Total: (200 items)			103,610.00	61,744.00	10,089.00	573,086.00
▲ Line: Washers (55 items)			27,656.00	16,865.00	2,697.00	144,544.00
▲ Color: Green (16 items)			8,150.00	4,658.00	669.00	33,853.00
▲ Price: Medium (1 items)			68.00	233.00	3.00	1,171.00
Washers	Green	P 0	68.00	233.00	3.00	1,171.00
▲ Price: Very High (8 items)			5,994.00	1,797.00	402.00	18,476.00
Washers	Green	P 2	678.00	201.00	12.00	2,748.00
Washers	Green	P 23	840.00	283.00	60.00	2,077.00
Washers	Green	P 42	687.00	107.00	59.00	1,856.00
Washers	Green	P 88	747.00	408.00	88.00	899.00
Washers	Green	P 91	630.00	249.00	82.00	3,505.00
Washers	Green	P 132	658.00	15.00	13.00	3,857.00
Washers	Green	P 187	889.00	349.00	68.00	2,952.00
Washers	Green	P 194	865.00	185.00	20.00	582.00
▲ Price: High (6 items)			2,084.00	2,476.00	193.00	12,436.00
Washers	Green	P 67	487.00	521.00	65.00	2,196.00
Washers	Green	P 75	257.00	350.00	19.00	526.00

FlexGrid comes with the CellFactory class to create every cell that appears on the grid. You can create custom cells by creating a class in your project that implements the ICellFactory interface and assign it to the CellFactory property of FlexGrid.

Custom ICellFactory classes can be highly specialized and application-specific, or can be very generic and reusable. In general, custom ICellFactory classes are simpler than custom columns since they deal directly with cells. Implementing custom **CellFactory** classes is fairly easy because you can inherit from the default CellFactory class included with the C1FlexGrid class. The default CellFactory class was designed to be extensible, so you can let it handle all the details of cell creation and customize only what you need.

The following image shows custom cells created through CellFactory in FlexGrid.

Symbol	Name	Bid	Ask	Last Sale	
A	Agilent Technologies	765.95 (2.9%) ▲ ∿	808.60 (4.5%) ▲ ⌒	787.27 (3.7%)	
AA	Alcoa Inc.	940.02 (-1.4%) ▼ ⌢	856.95 (3.2%) ▲ ⌒	898.49 (0.7%)	
AACC	Asset Acceptance Capital Corp.	712.93 (-3.0%) ▼ ⌒	678.27 (1.7%) ▲ ⌒	695.60 (-0.8%)	
AAME	Atlantic American Corporation	842.38 (4.9%) ▲ ⌒	980.37 (-2.6%) ▼ ⌒	911.38 (0.7%)	
AANB	Abigail Adams National Bancorp, Inc.	784.71 (0.3%) ▲ ∿	749.68 (-0.9%) ▼ ⌒	767.19 (-0.3%)	
AAON	AAON, Inc.	187.56 (6.0%) ▲ ⌣	198.61 (5.1%) ▲ ⌒	193.08 (5.5%)	
AAPL	Apple Inc.	31.14 (-2.0%) ▼ ⌒	30.88 (-1.6%) ▼ ⌣	31.01 (-1.8%)	
AATI	Advanced Analogic Technologies, Inc.	136.74 (2.5%) ▲ ⌣	144.40 (2.9%) ▲ ⌒	140.57 (2.7%)	
AAUK	Anglo American plc	168.06 (0.8%) ▲ ∿	138.89 (-3.7%) ▼ ⋀⋀	153.47 (-1.3%)	
AAWW	Atlas Air Worldwide Holdings	259.02 (5.6%) ▲ ⌣	197.86 (3.6%) ▲ ⋁⋁	228.44 (4.7%)	
ABAX	ABAXIS, Inc.	758.44 (5.6%) ▲ ⌒	687.37 (-2.1%) ▼ ⌒	722.91 (1.8%)	
ABBC	Abington Bancorp, Inc.	215.18 (-0.4%) ▼ ⌒	180.83 (-3.7%) ▼ ⌒	198.01 (-1.9%)	

When using custom cells, it is important to understand that grid cells are transient. Cells are constantly created and destroyed as the user scrolls, sorts, or selects ranges on the grid. This process is known as virtualization and is quite common in WPF applications. Without virtualization, a grid would typically have to create several thousand visual elements at the same time, which would impact its performance.

If you want more control over the printing process, use the GetPageImages method to automatically break up the grid into images that can be rendered onto individual pages. Each image is a 100% accurate representation of a portion of the grid, including styles, custom elements, repeating row and column headers on every page, and so on.

The **GetPageImages** method also allows callers to scale the images so the entire grid renders in actual size, scales to fit onto a single page, or scales to the width of a single page.

Once you have obtained the page images, you can use the WPF printing support to render them into documents with complete flexibility. For example, you can create documents that contain several grids, charts, and other types of content. You can also customize headers and footers, add letterheads, and so on.

The following sections demonstrate how an application can render the **FlexGrid** using the **GetPageImages** onto a print document in either platform.

Printing a C1FlexGrid in WPF

Printing documents in WPF requires the following steps:

1. Create a **PrintDialog** object.
2. If the dialog box's **ShowDialog** method returns true, then:
3. Create a **Paginator** object that will provide the document content.
4. Call the dialog's **Print** method.Row details template is a data panel that can be added to each row for displaying details. FlexGrid provides the flexibility to show information about each row through a template. You can embed text, UI elements, and data-bound controls, such as InputPanel, in the row details template. For each row, you can insert a data template to present its summary and show/provide details in other controls, such as text box, without affecting the dimensions of the grid. You can also use this template to create hierarchical grids displaying grouped data. In this example, we use row details template to display product-related information in FlexGrid.

The following image shows details of each row displayed through a row details template.

FlexGrid for WPF support ComponentOne's new ClearStyle technology that allows you to easily change control colors without having to change control templates. By just setting a few color properties you can quickly style the control.

Each key piece of the control's style is surfaced as a simple color property. This leads to a unique set of style properties for each control. For example, a **Gauge** has **PointerFill** and **PointerStroke** properties, whereas a **DataGrid** has **SelectedBrush** and **MouseOverBrush** for rows.

Let's say you have a control on your form that does not support ClearStyle. You can take the XAML resource created by ClearStyle and use it to help mold other controls on your form to match (such as grabbing exact colors). Or let's say you'd like to override part of a style set with ClearStyle (such as your own custom scrollbar). This is also possible because ClearStyle can be extended and you can override the style where desired.

ClearStyle is intended to be a solution to quick and easy style modification but you're still free to do it the old fashioned way with ComponentOne's controls to get the exact style needed. ClearStyle does not interfere with those less common situations where a full custom design is required.

You can completely change the appearance of the C1FlexGrid control by setting one or more properties, For example, if you set the **AlternatingRowBackground** property to "#FFC3F2F2", the C1FlexGrid control appears similar to the following:

Hover styles are applied to cells when the mouse hovers over the grid. It provides visual cues so that the user can apply styling to a cell, row, column or to a specific cell according to the GridSelectionMode enumeration when they are hovered on prior to selection or editing.

The hover styles can make the FlexGrid appear more "alive" and interactive.
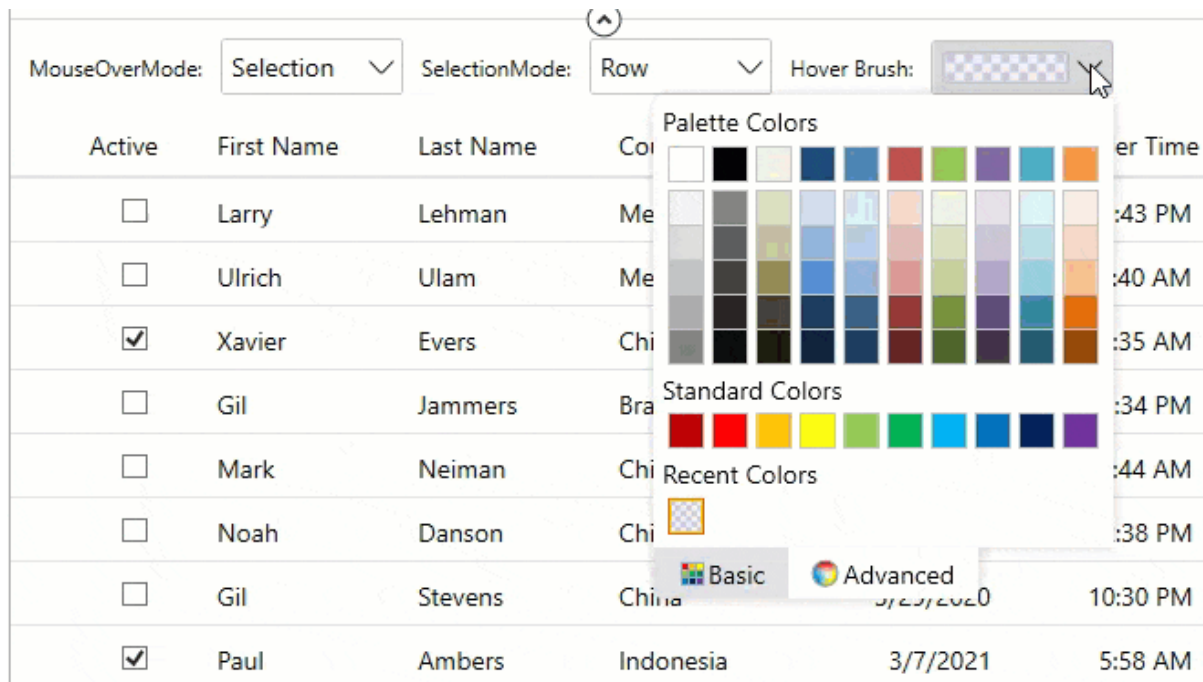
There are two essential properties to consider while performing hover styles, the **MouseOverMode** property and **MouseOverBrush** property in **GridBase** class. The MouseOverMode property helps to set how the mouse hovers over the grid, while the MouseOverBrush property sets a brush to paint the background of the selected cells. Also, **SelectionMode** property is equally important as it helps to set how the cells or rows are selected in the grid, since styling after all is applied based on cell hovering before selection. Moreover, the SelectionMode property calls the **GridSelectionMode** enumeration, the MouseOverMode property calls the **GridMouseOverMode** enumeration and the MouseOverBrush property calls the **Brush** class.

This feature in **WPF FlexGrid** allows you to freeze the rows and columns using mouse drag during runtime following a particular sequence. You can set the AllowFreezing Enum to **Columns** to freeze only columns, **Rows** to freeze only rows, or **Both** to freeze both columns and rows. Conversely, to disable freezing, set the AllowFreezing Enum to None, which is the default setting. This Enum can be set either in the designer or in code.

Implementation

In Designer

Locate the AllowFreezing Enum in the Properties window and set it to Rows, Columns, Both or None.

Now, you can manually use the mouse drag to adjust the number of frozen columns and rows as needed.

Feature Illustration

When the mouse pointer becomes the lock rows or the lock columns icon, click and drag the mouse over the rows or columns to freeze in a sequence. Setting the AllowFreezing Enum to **Both** allows both rows and columns to be frozen at the same time, as seen in the following GIF.

Id	First Name	Last Name	Address	City
0	Herb	Ulam	766 Park AVE	Saint Petersburg
1	Herb	Evers	569 Broad BLVD	Bekasi
2	Jack	Heath	24 Park BLVD	Hyderabad
3	Quince	Myers	378 Grand AVE	Yekaterinburg
4	Dan	Paulson	653 Park BLVD	Hyderabad
5	Mark	Orsted	171 Broad ST	Pune
6	Quince	Bishop	747 Main ST E	Ōsaka
7	Ed	Danson	30 Broad AVE	Fortaleza
8	Gil	Heath	117 Main ST	Delhi
9	Fred	Ulam	339 Green AVE	Chelyabinsk
10	Steve	Krause	141 Park BLVD	Yokohama
11	Noah	Stevens	915 Grand ST	Curitiba
12	Oprah	Cole	83 Broad ST	Sapporo
13	Mark	Krause	832 Main AVE	Gujranwala
14	Zeb	Ambers	14 Broad BLVD	Nagoya
15	Andy	Lehman	14 Golden BLVD	Mumbai
16	Oprah	Griswold	657 Green AVE	Fukuoka

<u>CHAPTER 5</u>

CODING

5.1 XAML Part

```xml
<UserControl x:Class="StoreSalesRegisterApplication.Pages.DashboardPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:StoreSalesRegisterApplication.Pages"
        xmlns:myuc="clr-namespace:StoreSalesRegisterApplication.UserControls"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <myuc:NavigationBar x:Name="navigationBar"></myuc:NavigationBar>
        <Label          Content="Dashboard"          HorizontalAlignment="Center"
VerticalAlignment="Center" Grid.Row="1" FontSize="24"/>
    </Grid>
</UserControl>
<UserControl x:Class="StoreSalesRegisterApplication.Pages.OrderPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```xml
        xmlns:local="clr-namespace:StoreSalesRegisterApplication.Pages"

        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"

        xmlns:myvm ="clr-namespace:StoreSalesRegisterApplication.ViewModel"

        xmlns:myuc="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">
<UserControl.Resources>
    <myvm:DataViewModel x:Key="dataViewModel"></myvm:DataViewModel>
</UserControl.Resources>
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>
    <myuc:NavigationBar x:Name="navigationBar"></myuc:NavigationBar>
    <Grid Grid.Row="1">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="147*"/>
            <ColumnDefinition Width="253*"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions >
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <Grid Grid.Row="0" Grid.ColumnSpan="2">
            <Grid.RowDefinitions >
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
                <RowDefinition Height="Auto"/>
```

```xml
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions >

            <ColumnDefinition Width="17*"/>

            <ColumnDefinition Width="Auto" MinWidth="271"/>

            <ColumnDefinition Width="Auto" MinWidth="89"/>

            <ColumnDefinition Width="38*"/>

        </Grid.ColumnDefinitions>

        <Label Margin="81,22,248,57" Grid.Row="1" Width="78" Content="Date" HorizontalAlignment="Center" Grid.ColumnSpan="2"/>

        <c1:C1DatePicker CustomFormat="dd-MM-yyyy" Grid.Column="1" Margin="0,22,63,57" Grid.Row="1" Width="180" c1:Name="userDate" HorizontalAlignment="Right" SelectedDateFormat="Long"/>

        <Label Grid.Column="2" Margin="20,22,0,57" Grid.Row="1" Width="69" Content="Time" HorizontalAlignment="Left"/>

        <StackPanel Grid.Column="3" Grid.Row="1" Margin="10,22,0,57" Orientation="Horizontal">

            <myuc:TimeHour IncreamentClick="HourIncreament" DecreamentClick="HourDecreament"></myuc:TimeHour>

            <c1:C1TimeEditor Width="100" ShowButtons="False" Name="userTime" Value="8:0:0" HorizontalAlignment="Left" Format="ShortTime"/>

            <myuc:TimeMinute IncreamentClick="MinuteIncreament" DecreamentClick="MinuteDecreament" AmPmEventClick="AmPmClick"></myuc:TimeMinute>

        </StackPanel>

        <Label Grid.Row="2" Margin="81,10,264,68" HorizontalAlignment="Center" Content="Location" RenderTransformOrigin="0.216,0.12" Width="62" Grid.ColumnSpan="2"/>

        <ComboBox Grid.Row="2" Grid.Column="1" Margin="0,10,63,68" Width="180" Name="locationComboBox" HorizontalAlignment="Right" SelectedIndex="0">

            <ComboBox.ItemTemplate>

              <DataTemplate>

                <TextBlock Text="{Binding Path=Name}"/>

              </DataTemplate>
```

```xml
          </ComboBox.ItemTemplate>

        </ComboBox>

      </Grid>

      <Grid Grid.Row="1" Grid.ColumnSpan="2">

        <Grid.ColumnDefinitions >

          <ColumnDefinition Width="*"/>

          <ColumnDefinition Width="*"/>

        </Grid.ColumnDefinitions>

        <StackPanel>

          <Label Content="Product" Margin="85,0,55,0" VerticalAlignment="Top"
FontSize="25" Height="42" BorderThickness="1" BorderBrush="Black"/>

          <ListBox Name="filterItems" Height="213" Margin="85,0,55,0">

            <ListBox.ItemTemplate>

              <DataTemplate>

                <StackPanel Orientation="Horizontal">

                  <Image Height="35" Width="35" Stretch="Fill" Source="{Binding
ImagePath}"></Image>

                  <CheckBox Grid.Column="1" Content="{Binding Path=Name}"
IsChecked="{Binding Source={StaticResource dataViewModel},Path=IsSelected,
Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" VerticalAlignment="Center"
Name="foodItemsCkBox" Checked="FoodItemChecked"
Unchecked="FoodItemUnchecked">

                    <CheckBox.LayoutTransform>

                      <ScaleTransform ScaleX="1.5" ScaleY="1.5"/>

                    </CheckBox.LayoutTransform>

                    <CheckBox.Style>

                      <Style TargetType="CheckBox">

                        <Setter Property="Foreground" Value="Red"></Setter>

                        <Style.Triggers>

                          <Trigger Property="IsChecked" Value="True">

                            <Setter Property="Foreground" Value="Green"/>
```

```xml
                    <Setter Property="FontWeight" Value="Bold"/>

                  </Trigger>

                </Style.Triggers>

              </Style>

            </CheckBox.Style>

            <CheckBox.Resources>

              <Style TargetType="Border">

                <Setter Property="CornerRadius" Value="3"/>

              </Style>

            </CheckBox.Resources>

          </CheckBox>

        </StackPanel>

      </DataTemplate>

    </ListBox.ItemTemplate>

  </ListBox>

</StackPanel>

<StackPanel      Grid.Column="1"      Margin="20,0,33,-103"      Height="300"
VerticalAlignment="Top">

    <ListBox    x:Name="selectedItems"    Height="200"    BorderThickness="0"
Margin="10,0,52,0">

      <ListBox.ItemTemplate>

        <DataTemplate>

          <StackPanel Orientation="Horizontal">

            <Image Source="{Binding ImagePath}" Width="80" Height="60"/>

            <StackPanel>

              <Label Content="{Binding Description}"/>

              <StackPanel Orientation="Horizontal">

                <Label Content="Enter Unit"/>

                <c1:C1NumericBox  Value="{Binding Units, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"   Minimum="1"   Maximum="500"   Width="50"
x:Name="inputUnitBox" ValueChanged="UnitChanged"/>
```

24

```xml
            </StackPanel>

          </StackPanel>

        </StackPanel>

      </DataTemplate>

    </ListBox.ItemTemplate>

  </ListBox>

  <Border BorderThickness="1" BorderBrush="Black" Margin="10,0,52,0">

    <Grid Margin="9,0">

      <Grid.ColumnDefinitions>

        <ColumnDefinition Width="Auto"/>

        <ColumnDefinition Width="*"/>

        <ColumnDefinition Width="Auto"/>

        <ColumnDefinition Width="*"/>

      </Grid.ColumnDefinitions>

      <Grid.RowDefinitions>

        <RowDefinition Height="*"/>

        <RowDefinition Height="*"/>

        <RowDefinition Height="*"/>

      </Grid.RowDefinitions>

      <Label Content="Total Items:"/>

      <Label          DataContext="{StaticResource          dataViewModel}"
Content="{Binding                NumberOfItems,                Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}"                    Grid.Column="1"
x:Name="selectedItemNumbers"   Width="100"   Height="25"   BorderBrush="Black"
BorderThickness="0,0,0,1" VerticalAlignment="Top" HorizontalAlignment="Left" />

      <Label Content="Total Units:" Grid.Column="2"/>

      <Label   Grid.Column="3"   x:Name="selectedItemUnits"   Content="0"
Width="95"      Height="25"      BorderBrush="Black"      BorderThickness="0,0,0,1"
VerticalAlignment="Top" HorizontalAlignment="Left" />

      <Label Content="Total Price:"  Grid.Row="1"/>
```

```xml
<Label Grid.Column="1" Grid.Row="1" x:Name="selectedItemPrice"
Content="0" Width="100" Height="25" BorderBrush="Black" BorderThickness="0,0,0,1"
VerticalAlignment="Top" HorizontalAlignment="Left" />

        </Grid>

    </Border>

    <StackPanel>

     <Grid>

      <Grid.ColumnDefinitions>

        <ColumnDefinition Width="130"/>

        <ColumnDefinition Width="*"/>

      </Grid.ColumnDefinitions>

      <Button x:Name="submitButton" ToolTip="Submit Data" Margin="10"
Width="100" Height="30" HorizontalAlignment="Left" Background="CornflowerBlue"
Content="Submit" FontWeight="Bold" Click="BillSubmit" OpacityMask="Black">

        <Button.Style>

          <Style TargetType="Button">

            <Style.Triggers>

              <Trigger Property="IsMouseOver" Value="True">

                <Trigger.EnterActions>

                  <BeginStoryboard>

                    <Storyboard>

                      <DoubleAnimation    Duration="0:0:0.1"    To="35"
Storyboard.TargetProperty="Height"/>

                      <DoubleAnimation    Duration="0:0:0.1"    To="110"
Storyboard.TargetProperty="Width"/>

                    </Storyboard>

                  </BeginStoryboard>

                </Trigger.EnterActions>

                <Trigger.ExitActions>

                  <BeginStoryboard>

                    <Storyboard>
```

```xml
                            <DoubleAnimation          Duration="0:0:0.1"          To="30"
Storyboard.TargetProperty="Height"/>
                                <DoubleAnimation          Duration="0:0:0.1"          To="100"
Storyboard.TargetProperty="Width"/>
                        </Storyboard>
                    </BeginStoryboard>
                </Trigger.ExitActions>
            </Trigger>
        </Style.Triggers>
    </Style>
</Button.Style>
<Button.Triggers>
    <EventTrigger RoutedEvent="Button.Click">
        <BeginStoryboard>
            <Storyboard>
                <DoubleAnimation                      BeginTime="0:0:0"
Storyboard.TargetName="lblStatus"      Storyboard.TargetProperty="FontSize"      From="1"
To="17" Duration="0:0:0.2" />
                <DoubleAnimation                      BeginTime="0:0:2.5"
Storyboard.TargetName="lblStatus"      Storyboard.TargetProperty="FontSize"      From="17"
To="0.1" Duration="0:0:0.2" />
            </Storyboard>
        </BeginStoryboard>
    </EventTrigger>
</Button.Triggers>
<Button.Resources>
    <Style TargetType="Border">
        <Setter Property="CornerRadius" Value="5"/>
    </Style>
</Button.Resources>
</Button>
```

```xml
                <Label    Grid.Column="1"    x:Name="lblStatus"    FontWeight="Bold"
FontSize="17" HorizontalAlignment="Center" VerticalAlignment="Center"/>
            </Grid>
          </StackPanel>
        </StackPanel>
      </Grid>
    </Grid>
  </Grid>
</UserControl>
<UserControl x:Class="StoreSalesRegisterApplication.Pages.ReportsPage"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:StoreSalesRegisterApplication.Pages"
        xmlns:mymv="clr-namespace:StoreSalesRegisterApplication.ViewModel"
        xmlns:myuc="clr-namespace:StoreSalesRegisterApplication.UserControls"
        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
  <UserControl.Resources>
    <mymv:DataViewModel x:Key="dataViewModel"/>
  </UserControl.Resources>
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
      <RowDefinition Height="45"/>
```

```xml
    </Grid.RowDefinitions>

    <myuc:NavigationBar x:Name="navigationBar"></myuc:NavigationBar>

    <Grid Grid.Row="1">

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="*"/>

            <ColumnDefinition Width="*"/>

        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>

            <RowDefinition Height="45"/>

            <RowDefinition Height="40"/>

        </Grid.RowDefinitions>

        <myuc:ColumnFilter        x:Name="columnFilter"        ItemFilterClick="ItemFilter"
ColumnFilterChanged="ColumnFilterUpdated"                        Margin="20,0,0,0"
Grid.ColumnSpan="2"></myuc:ColumnFilter>

        <StackPanel Grid.Column="1" Orientation="Horizontal">

            <ToggleButton  Content="Collapse All" Width="90" ToolTip="Collapse/Expand
the    current    view    of    data"    Margin="70,10,5,5"    Background="CornflowerBlue"
Name="groupRowVisibility" Checked="CollapseAll" Unchecked="ExpandAll">

                <ToggleButton.Resources>

                    <Style TargetType="Border">

                        <Setter Property="CornerRadius" Value="3"/>

                    </Style>

                </ToggleButton.Resources>

            </ToggleButton>

            <ToggleButton Content="Sort(Des) by Id" Name="sortById" ToolTip="Sort Data
of Grid by Order Id" Margin="10,10,5,5" Width="90" Checked="SortGridDescending"
Unchecked="SortGridAscending"  Background="CornflowerBlue">

                <ToggleButton.Resources>

                    <Style TargetType="Border">

                        <Setter Property="CornerRadius" Value="3"/>

                    </Style>
```

```
            </ToggleButton.Resources>

        </ToggleButton>

        <Button Content="Refresh" ToolTip="Refresh Grid with Stores Bill Data"
Margin="10,10,5,5" Width="90" Click="RefreshGrid" Background="CornflowerBlue">

            <Button.Resources>

              <Style TargetType="Border">

                <Setter Property="CornerRadius" Value="3"/>

              </Style>

            </Button.Resources>

        </Button>

    </StackPanel>

    <myuc:DateRange x:Name="gridDateRange" Grid.Row="1" Margin="20,0,0,0"
DateFilterEvent="DateFilter"></myuc:DateRange>

    <TextBlock Text="Search" FontSize="18" Grid.Row="1" Grid.Column="1"
Width="70" TextDecorations="Underline" Margin="-180,13,0,0"></TextBlock>

    <TextBox x:Name="searchFilter" TextChanged="searchTextChange" Grid.Row="1"
Grid.Column="1" Width="240" Height="25" Margin="100,10,0,0" FontSize="18">

        <TextBox.Resources>

          <Style TargetType="Border">

            <Setter Property="CornerRadius" Value="5"/>

          </Style>

        </TextBox.Resources>

    </TextBox>

  </Grid>

  <c1:C1FlexGrid Name="salesGrid" IsReadOnly="True" BorderThickness="0,0,0,1"
BorderBrush="Gray" AllowResizing="Both" AllowFreezing="Columns"

        ResizingColumn="ColumnResize"
ColumnHeaderBackground="CornflowerBlue"        SortedColumn="ColumnsSorted"
Grid.Row="2" ChildItemsPath="FoodItem"

        HeadersVisibility="Column" Margin="10" AutoGenerateColumns="False"
DataContext="{StaticResource        dataViewModel}"        TreeIndent="10"
VerticalScrollBarVisibility="Visible">
```

```xml
<c1:C1FlexGridFilterService.FlexGridFilter>

<c1:C1FlexGridFilter                    x:Name="columnLevelFilter"
FilterApplied="ColumnFilterApplied"></c1:C1FlexGridFilter>

</c1:C1FlexGridFilterService.FlexGridFilter>

<c1:C1FlexGridFilterService.FullTextFilterBehavior>

<c1:C1FullTextFilter        x:Name="searchFullText"        Delay="0:0:0.5"
Mode="WhileTyping" MatchNumbers="True" MatchCase="false"/>

</c1:C1FlexGridFilterService.FullTextFilterBehavior>

<c1:C1FlexGrid.Columns>

<c1:Column      Header="Id"      Binding="{Binding      Id}"      Width="33"
MaxWidth="50"></c1:Column>

<c1:Column            Header="Date"            Binding="{Binding            Date}"
Width="100"></c1:Column>

<c1:Column            Header="Time"            Binding="{Binding            Time}"
Width="58"></c1:Column>

<c1:Column           Header="Location"           Binding="{Binding           Location}"
Width="70"></c1:Column>

<c1:Column           Header="Item(s)"           Binding="{Binding           Name}"
Width="70"></c1:Column>

<c1:Column         Header="Unit         Price"         Binding="{Binding         Price}"
GroupAggregate="Sum" Width="75"></c1:Column>

<c1:Column            Header="Units"            Binding="{Binding            Units}"
Width="60"></c1:Column>

<c1:Column         Header="Item         Price"         Binding="{Binding         ItemPrice}"
Width="80"></c1:Column>

<c1:Column       Header="Total       Units"       Binding="{Binding       NumberOfUnits}"
Width="80" HorizontalAlignment="Left"></c1:Column>

<c1:Column Header="Total Price" Binding="{Binding TotalPrice}" Width="100"
HorizontalAlignment="Left" Format="{}Rs ./-"></c1:Column>

</c1:C1FlexGrid.Columns>

</c1:C1FlexGrid>

<StackPanel Grid.Row="3" Orientation="Horizontal">

<myuc:FlexPagination        x:Name="flexPaginator"        Margin="10,0,25,10"
FirstPageClick="FirstPageEvent"                    LastPageClick="LastPageEvent"
```

```
NextPageClick="NextPageEvent"                    PreviousPageClick="PreviousPageEvent"
RecordPerPageChanged="RecordPerPageEvent"></myuc:FlexPagination>

        <Button Content="Copy" ToolTip="Copy Selected Data" Click="CopySelectedData"
Margin="10,0,5,10" FontSize="15" Width="65" Background="CornflowerBlue">

            <Button.Resources>

                <Style TargetType="Border">

                    <Setter Property="CornerRadius" Value="3"/>

                </Style>

            </Button.Resources>

        </Button>

        <Button Content="Copy All" ToolTip="Copy All Data" Click="CopyAllData"
Margin="10,0,5,10" FontSize="15" Width="65" Background="CornflowerBlue">

            <Button.Resources>

                <Style TargetType="Border">

                    <Setter Property="CornerRadius" Value="3"/>

                </Style>

            </Button.Resources>

        </Button>

        <Button   Content="Preview"   ToolTip="Print   Preview   of   Grid   Data"
Click="PrintPreviewGrid"        Margin="10,0,5,10"        FontSize="15"        Width="65"
Background="CornflowerBlue">

            <Button.Resources>

                <Style TargetType="Border">

                    <Setter Property="CornerRadius" Value="3"/>

                </Style>

            </Button.Resources>

        </Button>

        <Button   Content="Print"   ToolTip="Print   Data   of   Grid"   Click="PrintGrid"
Margin="10,0,5,10" FontSize="15" Width="65" Background="CornflowerBlue">

            <Button.Resources>

                <Style TargetType="Border">
```

```
                <Setter Property="CornerRadius" Value="3"/>

            </Style>

        </Button.Resources>

    </Button>

</StackPanel>

        </Grid>

</UserControl>

<UserControl x:Class="StoreSalesRegisterApplication.UserControls.ColumnFilter"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">

    <Grid>

        <Grid.ColumnDefinitions>

            <ColumnDefinition Width="100"/>

            <ColumnDefinition Width="100"/>

            <ColumnDefinition Width="50"/>

        </Grid.ColumnDefinitions>

        <Grid.RowDefinitions>

            <RowDefinition Height="20"/>

            <RowDefinition Height="20"/>

        </Grid.RowDefinitions>

        <TextBlock Text="Column" TextDecorations="Underline" Margin="10,0,0,0"/>

        <TextBlock     Text="Item"     Grid.Column="1"     TextDecorations="Underline"
Margin="10,0,0,0"/>

        <ComboBox        Grid.Column="0        SelectedIndex="0"        Grid.Row="1"
Name="comboHeaders" SelectionChanged="ColumnFilterChange" Margin="10,0,0,0"/>
```

```xaml
    <ComboBox Grid.Column="1" SelectedIndex="0" Grid.Row="1" Name="comboItems"
Margin="10,0,0,0"/>

    <Button Grid.Column="2" Content="Filter" ToolTip="Filtering on column level"
Grid.Row="1" Click="ItemFilterEvent" Background="CornflowerBlue" Margin="10,0,0,0">

        <Button.Resources>

            <Style TargetType="Border">

                <Setter Property="CornerRadius" Value="5"/>

            </Style>

        </Button.Resources>

    </Button>

  </Grid>

</UserControl>

<UserControl x:Class="StoreSalesRegisterApplication.UserControls.DateRange"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"

        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">

  <Grid>

    <Grid.ColumnDefinitions>

      <ColumnDefinition Width="100"/>

      <ColumnDefinition Width="100"/>

      <ColumnDefinition Width="50"/>

    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>

      <RowDefinition Height="20"/>

      <RowDefinition Height="20"/>
```

```
    </Grid.RowDefinitions>

    <TextBlock Text="Start Date" TextDecorations="Underline" Margin="10,0,0,0"/>

    <c1:C1DatePicker    Grid.Row="1"    Margin="10,0,0,0"    DisplayDateStart="1,1,2010"
SelectedDateChanged="StartDateChanged"                CustomFormat="dd-MM-yyyy"
Name="startDateFilter"/>

    <TextBlock    Text="End    Date"    Grid.Column="1"    TextDecorations="Underline"
Margin="10,0,0,0"/>

    <c1:C1DatePicker        Grid.Row="1"        Grid.Column="1"        Margin="10,0,0,0"
CustomFormat="dd-MM-yyyy" Name="endDateFilter"/>

    <Button    Grid.Column="2"    Content="Go"    ToolTip="Filtering    by    date    range"
Grid.Row="1" Margin="10,0,0,0" Click="DateFilterClick" Background="CornflowerBlue">

        <Button.Resources>

            <Style TargetType="Border">

                <Setter Property="CornerRadius" Value="5"/>

            </Style>

        </Button.Resources>

    </Button>

  </Grid>

</UserControl>

<UserControl x:Class="StoreSalesRegisterApplication.UserControls.FlexPagination"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">

  <Grid>

    <Grid.ColumnDefinitions>

        <ColumnDefinition Width="50"></ColumnDefinition>

        <ColumnDefinition Width="50"></ColumnDefinition>
```

```xml
            <ColumnDefinition Width="Auto"></ColumnDefinition>

            <ColumnDefinition Width="50"></ColumnDefinition>

            <ColumnDefinition Width="50"></ColumnDefinition>

            <ColumnDefinition Width="Auto"/>

        </Grid.ColumnDefinitions>

        <Button Click="FirstPage" Name="btnFirstPage" ToolTip="Go to first page" FontSize="24" Background="CornflowerBlue" Margin="5,0,0,0">

            <Border>

                <StackPanel Orientation="Horizontal">

                    <TextBlock Text="&#9615;" Margin="0,-6,0,0" Height="25"/>

                    <TextBlock Text="&#9204;" Margin="-7,-3,0,0"/>

                </StackPanel>

            </Border>

            <Button.Resources>

                <Style TargetType="Border">

                    <Setter Property="CornerRadius" Value="3"/>

                </Style>

            </Button.Resources>

        </Button>

        <Button Grid.Column="1" Name="btnPreviousPage" ToolTip="Go to previous page" FontSize="24" Click="PreviousPage" Background="CornflowerBlue" Margin="5,0,0,0">

            <TextBlock Text="&#9204;" Margin="0,-3,0,0"/>

            <Button.Resources>

                <Style TargetType="Border">

                    <Setter Property="CornerRadius" Value="3"/>

                </Style>

            </Button.Resources>

        </Button>

        <StackPanel Grid.Column="2" Orientation="Horizontal" Width="Auto">

            <Label Content="Page" FontSize="18" Width="Auto"></Label>
```

```xml
<Label Name="lblCurrentPage"  FontSize="18" Width="Auto"></Label>

<Label Content="of" FontSize="18" Width="Auto"></Label>

<Label Name="lblTotalPage" FontSize="18" Width="Auto"></Label>

</StackPanel>

<Button  Grid.Column="3"  Name="btnNextPage"  ToolTip="Go  to  next  page"
Click="NextPage" FontSize="24" Background="CornflowerBlue" Margin="5,0,0,0">

<TextBlock Text="&#9205;" Margin="0,-3,0,0"/>

<Button.Resources>

  <Style TargetType="Border">

    <Setter Property="CornerRadius" Value="3"/>

  </Style>

</Button.Resources>

</Button>

<Button  Grid.Column="4"  Name="btnLastPage"  ToolTip="Go  to  last  page"
Click="LastPage" FontSize="24" Background="CornflowerBlue" Margin="5,0,0,0">

<Border>

  <StackPanel Orientation="Horizontal">

    <TextBlock Text="&#9205;" Margin="0,-3,0,0"/>

    <TextBlock Text="&#9615;" Margin="-9,-6,0,0" Height="25"/>

  </StackPanel>

</Border>

<Button.Resources>

  <Style TargetType="Border">

    <Setter Property="CornerRadius" Value="3"/>

  </Style>

</Button.Resources>

</Button>

<StackPanel Grid.Column="5" Margin="5,-5,0,0">

<TextBlock Text="Records per Page"/>
```

```xml
        <ComboBox Name="comboRecPerPage" ToolTip="Select Record per Page"
SelectedIndex="0" SelectionChanged="RecordPerPage"/>

    </StackPanel>

  </Grid>

</UserControl>

<UserControl x:Class="StoreSalesRegisterApplication.UserControls.NavigationBar"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">

    <UserControl.Background>

      <SolidColorBrush Color="CornflowerBlue"/>

    </UserControl.Background>

    <Grid>

      <Grid.RowDefinitions>

        <RowDefinition Height="5"/>

        <RowDefinition Height="35"/>

      </Grid.RowDefinitions>

      <Grid.ColumnDefinitions>

        <ColumnDefinition Width="20"/>

        <ColumnDefinition Width="250"/>

        <ColumnDefinition Width="250"/>

        <ColumnDefinition Width="250"/>

        <ColumnDefinition />

      </Grid.ColumnDefinitions>
```

```xml
            <Button Content="Order" FontWeight="Bold" FontSize="18" Name="OrderNavigation"
Background="Transparent" BorderBrush="Transparent" BorderThickness="0" Grid.Row="1"
Grid.Column="1" Click="OrderClick"/>

            <Button          Content="Reports"          FontWeight="Bold"          FontSize="18"
Name="ReportsNavigation"         Background="Transparent"         BorderBrush="Transparent"
BorderThickness="0" Grid.Row="1" Grid.Column="2" Click="ReportsClick" />

            <Button          Content="Dashboard"          FontWeight="Bold"          FontSize="18"
Name="DashboardNavigation"      Background="Transparent"      BorderBrush="Transparent"
BorderThickness="0" Grid.Row="1" Grid.Column="3" Click="DashboardClick" />

        </Grid>

</UserControl>

<UserControl x:Class="StoreSalesRegisterApplication.UserControls.TimeHour"

        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"

        mc:Ignorable="d"

        d:DesignHeight="450" d:DesignWidth="800">

    <Grid>

      <Grid.RowDefinitions>

        <RowDefinition Height="15"></RowDefinition>

        <RowDefinition Height="15"></RowDefinition>

      </Grid.RowDefinitions>

      <Button Click="HourIncreament" >

        <TextBlock Text="&#9206;" Margin="0,-3,0,0"/>

        <Button.Resources>

          <Style TargetType="Border">

            <Setter Property="CornerRadius" Value="3"/>

          </Style>

        </Button.Resources>
```

```xml
    </Button>
    <Button Grid.Row="1" Click="HourDecreament">
      <TextBlock Text="&#9207;" Margin="0,-3,0,0"/>
      <Button.Resources>
        <Style TargetType="Border">
          <Setter Property="CornerRadius" Value="3"/>
        </Style>
      </Button.Resources>
    </Button>
  </Grid>
</UserControl>
<UserControl x:Class="StoreSalesRegisterApplication.UserControls.TimeMinute"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:local="clr-namespace:StoreSalesRegisterApplication.UserControls"
        mc:Ignorable="d"
        d:DesignHeight="450" d:DesignWidth="800">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="15"></RowDefinition>
      <RowDefinition Height="15"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto"/>
      <ColumnDefinition Width="Auto"/>
    </Grid.ColumnDefinitions>
    <Button Click="MinuteIncreament">
```

```xml
<TextBlock Text="&#9206;" Margin="0,-3,0,0"/>

<Button.Resources>

  <Style TargetType="Border">

    <Setter Property="CornerRadius" Value="3"/>

  </Style>

</Button.Resources>

</Button>

<Button Grid.Row="1" Click="MinuteDecreament">

<TextBlock Text="&#9207;" Margin="0,-3,0,0"/>

<Button.Resources>

  <Style TargetType="Border">

    <Setter Property="CornerRadius" Value="3"/>

  </Style>

</Button.Resources>

</Button>

<Button Grid.Column="1" Grid.RowSpan="2" Content="AM/PM" Click="AmPmClick">

  <Button.Resources>

    <Style TargetType="Border">

      <Setter Property="CornerRadius" Value="3"/>

    </Style>

  </Button.Resources>

</Button>

</Grid>
```

5.2 CS Part

```csharp
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;


namespace StoreSalesRegisterApplication.Pages
{
    /// <summary>
    /// Interaction logic for DashboardPage.xaml
    /// </summary>
    public partial class DashboardPage : UserControl
    {
        public DashboardPage()
        {
            InitializeComponent();

            navigationBar.DashboardNavigation.Background = Brushes.White;
        }
    }
}
```

```csharp
using C1.WPF;

using Newtonsoft.Json;

using StoreSalesRegisterApplication.ViewModel;

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Animation;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;


namespace StoreSalesRegisterApplication.Pages

{

    /// <summary>

    /// Interaction logic for OrderPage.xaml

    /// </summary>

    public partial class OrderPage : UserControl

    {
```

```
DataViewModel dataVM;

public OrderPage()

{

    InitializeComponent();

    dataVM = new DataViewModel();

    DataContext = dataVM;

    LoadData();

}


/// <summary>

/// Load Controls

/// </summary>

private void LoadData()

{

    navigationBar.OrderNavigation.Background = Brushes.White;

    locationComboBox.ItemsSource = dataVM.Location;

    filterItems.ItemsSource = dataVM.FoodItems;

    selectedItems.ItemsSource = dataVM.selectedFoodItemList;

}

/// <summary>

/// Food Item Select

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FoodItemChecked(object sender, RoutedEventArgs e)

{

    CheckBox checkedBox = sender as CheckBox;

    FoodItem context = (FoodItem)checkedBox.DataContext;
```

```csharp
    if (context != null)

    {

        dataVM.AddItem(context);

        BillUpdate();

    }

}

/// <summary>

/// Food Item Unselect

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void FoodItemUnchecked(object sender, RoutedEventArgs e)

{

    CheckBox checkedBox = sender as CheckBox;

    FoodItem context = (FoodItem)checkedBox.DataContext;

    if (context != null)

    {

        dataVM.RemoveItem(context);

        BillUpdate();

    }

}

/// <summary>

/// Unit updation

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void UnitChanged(object sender, PropertyChangedEventArgs<double> e)

{

    C1NumericBox numericBox = sender as C1NumericBox;
```

```csharp
FoodItem context = (FoodItem)numericBox.DataContext;

if (context != null)

{

    BillUpdate();

}

}

private void BillUpdate()

{

    selectedItemUnits.Content = dataVM.NumberOfUnits.ToString();

    selectedItemPrice.Content = dataVM.TotalPrice.ToString();

}

/// <summary>

/// Submit Data

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void BillSubmit(object sender, RoutedEventArgs e)

{

    BillData bill = new BillData();

    Location location = locationComboBox.SelectedItem as Location;

    ObservableCollection<BillData> billData = new ObservableCollection<BillData>();

    if(File.Exists(Properties.Resources.BillDataFile))

        billData = JsonConvert.DeserializeObject<ObservableCollection<BillData>>(new
FileDataOperation().ReadDataFromJson(Properties.Resources.BillDataFile));

    foreach(var item in dataVM.selectedFoodItemList)

    {

        FoodItemDetail foodItem = new FoodItemDetail();

        foodItem.Name = item.Name;

        foodItem.Units = item.Units;
```

```csharp
            foodItem.Price = item.Price;

            dataVM.SelectedFoodItemDetails.Add(foodItem);

        }

    bill.Id = billData.Count+1;

    bill.Date = userDate.Text;

    bill.Time = userTime.Value.ToString();

    bill.Location = location.Name;

    bill.FoodItem = dataVM.SelectedFoodItemDetails;

    bill.NumberOfItems = Convert.ToDouble(selectedItemNumbers.Content.ToString());

    bill.NumberOfUnits = Convert.ToDouble(selectedItemUnits.Content.ToString());

    bill.TotalPrice = Convert.ToDouble(selectedItemPrice.Content.ToString());

    if (bill.NumberOfItems > 0)

    {

        billData.Add(bill);

        string filePath = Properties.Resources.BillDataFile;

        string fileContent = JsonConvert.SerializeObject(billData);

        new FileDataOperation().WriteDataToJson(filePath, fileContent);

        lblStatus.Content = Properties.Resources.SuccessMessage;

        lblStatus.Foreground = Brushes.Green;

    }

    else

    {

        lblStatus.Content = Properties.Resources.LessItemSelection;

        lblStatus.Foreground = Brushes.Red;

    }

}


private void HourIncreament(object sender, RoutedEventArgs e)

{
```

```csharp
            userTime.Value += new TimeSpan(1, 0, 0);
        }


        private void HourDecreament(object sender, RoutedEventArgs e)
        {
            userTime.Value -= new TimeSpan(1, 0, 0);
        }


        private void MinuteIncreament(object sender, RoutedEventArgs e)
        {
            userTime.Value += new TimeSpan(0, 1, 0);
        }


        private void MinuteDecreament(object sender, RoutedEventArgs e)
        {
            userTime.Value -= new TimeSpan(0, 1, 0);
        }


        private void AmPmClick(object sender, RoutedEventArgs e)
        {
            if(userTime.Value<=new TimeSpan(12,0,0))
                userTime.Value += new TimeSpan(12, 0, 0);
            else
                userTime.Value -= new TimeSpan(12, 0, 0);


        }
    }
}
```

```csharp
using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

using C1.WPF.FlexGrid;

using StoreSalesRegisterApplication.ViewModel;


namespace StoreSalesRegisterApplication.Pages
{
    /// <summary>
    /// Interaction logic for ReportsPage.xaml
    /// </summary>
    public partial class ReportsPage : UserControl
    {
        DataViewModel dataVM;
        List<string> headerName;
        List<BillData> listView;
        List<BillData> listData;
```

```
int currentPage, recPerPage,totalPage;

public static string searchData;

public ReportsPage()

{

  InitializeComponent();

  dataVM = new DataViewModel();

  headerName = new List<string>();

  navigationBar.ReportsNavigation.Background = Brushes.White;

  listData = new List<BillData>();

  listView = new List<BillData>();

  listData = dataVM.BillOrder.ToList();

  LoadReports();

}

/// <summary>

/// Reports Tab Load

/// </summary>

private void LoadReports()

{

  if (new DataViewModel().BillOrder.Count < 1)

    MessageBox.Show(Properties.Resources.EmptyGridMessage,
Properties.Resources.EmptyGridCaption,                          MessageBoxButton.OK,
MessageBoxImage.Warning);

  RefreshGrid(new object(), new RoutedEventArgs());

  salesGrid.CellFactory = new GridCellStyle();

  new C1FlexGridFilter(salesGrid);


  foreach (var column in salesGrid.Columns)

    headerName.Add(column.Header);

  columnFilter.comboHeaders.ItemsSource = headerName;

  searchFullText.FilterEntry = searchFilter;
```

```
        salesGrid.ColumnHeaders.MouseUp += ColumnHeaderClick;
    }

/// <summary>

/// column filter updation event for column data load

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void ColumnFilterUpdated(object sender, RoutedEventArgs e)

{

    int column = 0;

    foreach (var col in salesGrid.Columns)

    {

        if (col.Header == columnFilter.comboHeaders.SelectedItem.ToString())

        {

            column = col.Index;

            break;

        }

    }

    List<string> itemList = new List<string>();

    salesGrid.ItemsSource = dataVM.BillOrder;

    foreach (var row in salesGrid.Rows)

    {

        if (salesGrid.Cells[row.Index, column] != null)

            itemList.Add(salesGrid.Cells[row.Index, column].ToString());

        itemList = itemList.Distinct().ToList();

    }

    salesGrid.ItemsSource = listView;

    GroupRowDiffer();

    columnFilter.comboItems.ItemsSource = itemList;
```

```
            columnFilter.comboItems.SelectedIndex = 0;

        }

        /// <summary>
        /// column filter event
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void ItemFilter(object sender, RoutedEventArgs e)
        {
            bool found = false;
            int column = 0;
            foreach (var col in salesGrid.Columns)
            {
                if (col.Header == columnFilter.comboHeaders.SelectedItem.ToString())
                {
                    column = col.Index;
                    break;
                }
            }
            salesGrid.ItemsSource = dataVM.BillOrder;
            //for parent items
            if (salesGrid.Cells[0, column] != null)
            {
                foreach (var row in salesGrid.Rows)
                {
                    if (salesGrid.Cells[row.Index, column] != null && salesGrid.Cells[row.Index,
column].ToString().ToLower() != columnFilter.comboItems.Text.ToLower())
                    {
                        found = true;
```

```
                ((GroupRow)row).Visible = false;

                continue;

            }

            if (((GroupRow)row).HasChildren == true)

                found = false;

            if (found == true)

                ((GroupRow)row).Visible = false;

        }

    }

    //for child items

    else

    {

        for (int rowindex = salesGrid.Rows.Count - 1; rowindex >= 0; rowindex--)

        {

            if (salesGrid.Cells[rowindex, column] != null && salesGrid.Cells[rowindex,
column].ToString().ToLower() != columnFilter.comboItems.Text.ToLower())

            {

                ((GroupRow)salesGrid.Rows[rowindex]).Visible = false;

                continue;

            }

            if (((GroupRow)salesGrid.Rows[rowindex]).HasChildren == false)

                found = true;

            else

            {

                if (found == true)

                    found = false;

                else

                    ((GroupRow)salesGrid.Rows[rowindex]).Visible = false;

            }
```

```
        }

        //for remaining child items

        bool parentFound = false;

        foreach (var row in salesGrid.Rows)

        {

            if (((GroupRow)row).HasChildren)

            {

                if (((GroupRow)row).IsVisible)

                    parentFound = true;

                else

                    parentFound = false;

                continue;

            }

            if (parentFound == true)

                ((GroupRow)row).Visible = true;

        }

    }

    listData.Clear();

    foreach (var row in salesGrid.Rows)

    {

        if (((GroupRow)row).HasChildren && ((GroupRow)row).IsVisible)

        {

            listData.Add((BillData)((GroupRow)row).DataItem);

        }

    }

    FirstPageEvent(new object(), new RoutedEventArgs());

}

/// <summary>

/// Color differentiator for Group Rows
```

```csharp
/// </summary>
private void GroupRowDiffer()
{
    bool groupType = true;
    foreach (var row in salesGrid.Rows)
    {
        ((GroupRow)row).IsCollapsed = true;
        if (row.IsVisible == true)
            groupType = (!groupType);
        row.Background = groupType ? Brushes.White : Brushes.LightGray;
    }
    foreach (var row in salesGrid.Rows)
        ((GroupRow)row).IsCollapsed = false;
}
/// <summary>
/// Event for column header click
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ColumnHeaderClick(object sender, MouseButtonEventArgs e)
{
    salesGrid.ItemsSource = listData;
    GroupRowDiffer();
}


/// <summary>
/// Column Sort Event
/// </summary>
/// <param name="sender"></param>
```

```csharp
/// <param name="e"></param>
private void ColumnsSorted(object sender, CellRangeEventArgs e)
{
    listData.Clear();

    foreach(var row in salesGrid.Rows)
    {
        if ((((GroupRow)row).HasChildren && ((GroupRow)row).IsVisible)
        {
            listData.Add((BillData)((GroupRow)row).DataItem);
        }
    }

    FirstPageEvent(new object(), new RoutedEventArgs());

    GroupRowDiffer();

    ExpandAll(this,new RoutedEventArgs());
}
/// <summary>
/// Default column filter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ColumnFilterApplied(object sender, EventArgs e)
{
    GroupRowDiffer();

    ExpandAll(new object(), new RoutedEventArgs());
}
/// <summary>
/// Collapse Grid Rows
/// </summary>
/// <param name="sender"></param>
```

```csharp
/// <param name="e"></param>
private void CollapseAll(object sender, RoutedEventArgs e)
{
    groupRowVisibility.Content = Properties.Resources.GroupRowExpand;
    groupRowVisibility.IsChecked = true;
    foreach (var row in salesGrid.Rows)
        if (((GroupRow)row).IsVisible)
            ((GroupRow)row).IsCollapsed = true;
}
/// <summary>
/// Expand Grid Row
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ExpandAll(object sender, RoutedEventArgs e)
{
    groupRowVisibility.Content = Properties.Resources.GroupRowCollapse;
    groupRowVisibility.IsChecked = false;
    foreach (var row in salesGrid.Rows)
        if(((GroupRow)row).IsVisible)
            ((GroupRow)row).IsCollapsed = false;
}
/// <summary>
/// Sort grid by Id(Descending)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SortGridDescending(object sender, RoutedEventArgs e)
{
```

```csharp
    sortById.Content = Properties.Resources.SortAscending;

    sortById.IsChecked = true;

    listData = listData.OrderByDescending(o => o.Id).ToList();

    FirstPageEvent(new object(), new RoutedEventArgs());

}
/// <summary>
/// Sort grid by Id(Ascending)
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void SortGridAscending(object sender, RoutedEventArgs e)

{

    sortById.Content = Properties.Resources.SortDescending;

    sortById.IsChecked = false;

    listData = listData.OrderBy(o => o.Id).ToList();

    FirstPageEvent(new object(), new RoutedEventArgs());

}
/// <summary>
/// Refresh grid to the overall data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void RefreshGrid(object sender, RoutedEventArgs e)

{

    currentPage = 1;

    recPerPage = Convert.ToInt32(flexPaginator.comboRecPerPage.Text);

    listData = dataVM.BillOrder.ToList();

    totalPage = (listData.Count) / recPerPage;

    if ((listData.Count) % recPerPage != 0)
```

```csharp
        totalPage++;

    listView = listData.Take(recPerPage).ToList();

    salesGrid.ItemsSource = listView;

    flexPaginator.lblCurrentPage.Content = currentPage;

    flexPaginator.lblTotalPage.Content = totalPage;

    flexPaginator.btnFirstPage.IsEnabled = false;

    flexPaginator.btnPreviousPage.IsEnabled = false;

    flexPaginator.btnLastPage.IsEnabled = true;

    flexPaginator.btnNextPage.IsEnabled = true;

    GroupRowDiffer();

}

/// <summary>

/// Date Range Filter

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void DateFilter(object sender, RoutedEventArgs e)

{

    DateTime startDate = DateTime.Parse(gridDateRange.startDateFilter.Text);

    DateTime endDate = DateTime.Parse(gridDateRange.endDateFilter.Text);

    listData.Clear();

    foreach (var item in dataVM.BillOrder)

    {

        DateTime compareDate = DateTime.Parse(item.Date);

        if (compareDate > startDate && compareDate < endDate)

            listData.Add(item);

        if (compareDate == startDate || compareDate == endDate)

            listData.Add(item);

    }
```

```
            FirstPageEvent(new object(), new RoutedEventArgs());

            if(listData.Count==0)

                MessageBox.Show(Properties.Resources.EmptyGridMessage,
Properties.Resources.EmptyGridCaption,                              MessageBoxButton.OK,
MessageBoxImage.Warning);

        }

        /// <summary>

        /// column resize for trimming

        /// </summary>

        /// <param name="sender"></param>

        /// <param name="e"></param>

        private void ColumnResize(object sender, CellRangeEventArgs e)

        {

            salesGrid.Columns[e.Column].TextTrimming = TextTrimming.CharacterEllipsis;

            salesGrid.Columns[e.Column].HeaderTextTrimming                      =
TextTrimming.CharacterEllipsis;

        }

        /// <summary>

        /// navigate to first page

        /// </summary>

        /// <param name="sender"></param>

        /// <param name="e"></param>

        private void FirstPageEvent(object sender, RoutedEventArgs e)

        {

            currentPage = 1;

            recPerPage = Convert.ToInt32(flexPaginator.comboRecPerPage.Text);

            totalPage = (listData.Count) / recPerPage;

            if ((listData.Count) % recPerPage != 0)

                totalPage++;

            listView = listData.Take(recPerPage).ToList();
```

```csharp
        salesGrid.ItemsSource = listView;

        flexPaginator.lblCurrentPage.Content = currentPage;

        flexPaginator.lblTotalPage.Content = totalPage;

        flexPaginator.btnFirstPage.IsEnabled = false;

        flexPaginator.btnPreviousPage.IsEnabled = false;

        flexPaginator.btnLastPage.IsEnabled = true;

        flexPaginator.btnNextPage.IsEnabled = true;

        GroupRowDiffer();

    }
    /// <summary>
    /// Navigate to last page
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void LastPageEvent(object sender, RoutedEventArgs e)
    {
        currentPage = totalPage;

        listView = listData.Skip((currentPage-1) * recPerPage).Take(recPerPage).ToList();

        salesGrid.ItemsSource = listView;

        flexPaginator.lblCurrentPage.Content = currentPage;

        flexPaginator.btnFirstPage.IsEnabled = true;

        flexPaginator.btnPreviousPage.IsEnabled = true;

        flexPaginator.btnLastPage.IsEnabled = false;

        flexPaginator.btnNextPage.IsEnabled = false;

        GroupRowDiffer();

    }


    /// <summary>
    /// Navigate to next page
```

```csharp
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void NextPageEvent(object sender, RoutedEventArgs e)
{
    if(currentPage<totalPage)
    {
        listView = listData.Skip(currentPage * recPerPage).Take(recPerPage).ToList();
        salesGrid.ItemsSource = listView;
        GroupRowDiffer();
        currentPage++;
        flexPaginator.lblCurrentPage.Content = currentPage;
        flexPaginator.btnFirstPage.IsEnabled = true;
        flexPaginator.btnPreviousPage.IsEnabled = true;
    }
    if(currentPage == totalPage)
    {
        flexPaginator.btnLastPage.IsEnabled = false;
        flexPaginator.btnNextPage.IsEnabled = false;
    }
}

/// <summary>
/// Navigate to previous page
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void PreviousPageEvent(object sender, RoutedEventArgs e)
{
```

```
        if (currentPage > 1)

        {

            listView = listData.Skip((currentPage-2) * recPerPage).Take(recPerPage).ToList();

            salesGrid.ItemsSource = listView;

            GroupRowDiffer();

            currentPage--;

            flexPaginator.lblCurrentPage.Content = currentPage;

            flexPaginator.btnLastPage.IsEnabled = true;

            flexPaginator.btnNextPage.IsEnabled = true;

        }

        if (currentPage == 1)

        {

            flexPaginator.btnFirstPage.IsEnabled = false;

            flexPaginator.btnPreviousPage.IsEnabled = false;

        }

    }

    /// <summary>

    /// Record per Page updated

    /// </summary>

    /// <param name="sender"></param>

    /// <param name="e"></param>

    private void RecordPerPageEvent(object sender, RoutedEventArgs e)

    {

        flexPaginator.comboRecPerPage.Text                                =
flexPaginator.comboRecPerPage.SelectedItem.ToString();

        FirstPageEvent(this, new RoutedEventArgs());

    }

    /// <summary>

    /// Copy selected data from grid
```

```csharp
/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void CopySelectedData(object sender, RoutedEventArgs e)

{

    salesGrid.Copy();

    MessageBox.Show(Properties.Resources.Copy);

}

/// <summary>

/// Copy all Bill data

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void CopyAllData(object sender, RoutedEventArgs e)

{

    salesGrid.ItemsSource = dataVM.BillOrder;

    salesGrid.SelectAll();

    salesGrid.Copy();

    FirstPageEvent(new object(), new RoutedEventArgs());

    salesGrid.Select(0, 0);

    MessageBox.Show(Properties.Resources.Copy);

}


private void searchTextChange(object sender, TextChangedEventArgs e)

{

    salesGrid.ItemsSource = listData;

    if (((TextBox)sender).Text==string.Empty)

        FirstPageEvent(new object(), new RoutedEventArgs());

    searchData = ((TextBox)sender).Text;
```

```csharp
        }


        /// <summary>
        /// Print Preview of grid
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PrintPreviewGrid(object sender, RoutedEventArgs e)
        {
            salesGrid.ItemsSource = listData;

            GroupRowDiffer();

            salesGrid.PrintPreview(string.Empty, ScaleMode.ActualSize, new Thickness(0), 100);

            FirstPageEvent(new object(), new RoutedEventArgs());
        }

        /// <summary>
        /// Print contents of Grid
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void PrintGrid(object sender, RoutedEventArgs e)
        {
            salesGrid.ItemsSource = listData;

            GroupRowDiffer();

            PrintParameters printParameters = new PrintParameters();

            printParameters.Margin = new Thickness(20, 50, 20, 50);

            printParameters.ScaleMode = ScaleMode.ActualSize;

            salesGrid.Print(printParameters);

            FirstPageEvent(new object(), new RoutedEventArgs());
        }
```

```csharp
        }


        /// <summary>
        /// class for highlighting cell style
        /// </summary>
        public class GridCellStyle : CellFactory
        {
            public override void ApplyCellStyles(C1FlexGrid grid, CellType cellType, CellRange range, Border border)
            {
                var columnIndex = range.Column;
                var rowIndex = range.Row;
                int column = 0;
                foreach (var col in grid.Columns)
                {
                    if (col.Header == Properties.Resources.TimeFilterColumn)
                    {
                        column = col.Index;
                        break;
                    }
                }
                if(cellType==CellType.Cell)
                {
                    //group row style
                    if (((GroupRow)grid.Rows[rowIndex]).HasChildren)
                    {
                        border.BorderBrush = Brushes.Black;
                        border.BorderThickness = new Thickness(0, 1, 0, 0);
                    }
```

```csharp
            //time exceed cell style

            if (grid[rowIndex, columnIndex] != null)

            {

                if ((columnIndex == column) && (DateTime.Parse(grid.Cells[rowIndex,
columnIndex].ToString()) >= DateTime.Parse(new TimeSpan(21, 0, 0).ToString())))

                {

                    border.BorderBrush = Brushes.Red;

                    border.BorderThickness = new Thickness(3);

                    border.CornerRadius = new CornerRadius(3);

                }

                if       (ReportsPage.searchData!=null        &&         grid.Cells[rowIndex,
columnIndex].ToString().ToLower().Contains(ReportsPage.searchData.ToLower()))

                {

                    border.Background = Brushes.Yellow;

                }

            }

        }

    }

}
```

CHAPTER 6

DATA FLOW DIAGRAM

A data flow diagram (DFD) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system without going into great detail, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, how the data will advance through the system, and where the data will be stored. It does not show information about process timing or whether processes will operate in sequence or in parallel, unlike a traditional structured flowchart which focuses on control flow, or a UML activity workflow diagram, which presents both control and data flows as a unified model.

A picture is worth a thousand words. A Data Flow Diagram (DFD) is a traditional way to visualize the information flows within a system. A neat and clear DFD can depict a good amount of the system requirements graphically. It can be manual, automated, or a combination of both.

It shows how information enters and leaves the system, what changes the information and where information is stored. The purpose of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communications tool between a systems analyst and any person who plays a part in the system that acts as the starting point for redesigning a system.

It is usually beginning with a context diagram as level 0 of the DFD diagram, a simple representation of the whole system. To elaborate further from that, we drill down to a level 1 diagram with lower-level functions decomposed from the major functions of the system. This could continue to evolve to become a level 2 diagram when further analysis is required. Progression to levels 3, 4 and so on is possible but anything beyond level 3 is not very common. Please bear in mind that the level of detail for decomposing a particular function depending on the complexity that function.

CHAPTER 7

GANTT CHART
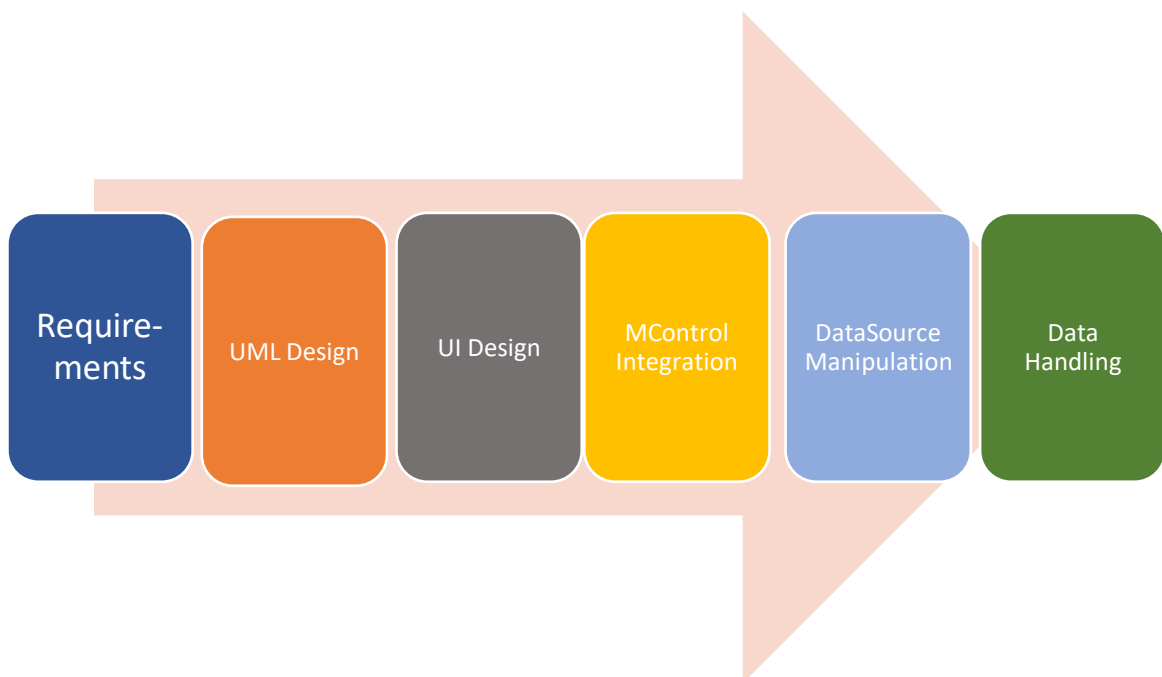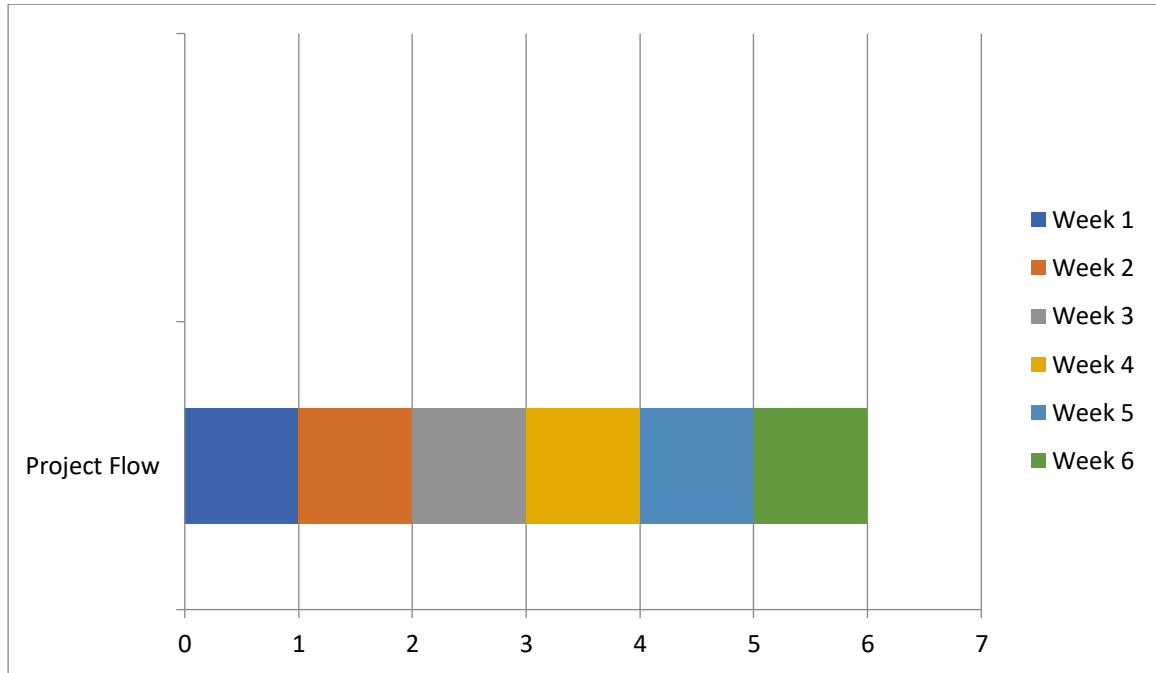
CHAPTER 8

FEASIBILITY STUDY

As the name implies, a feasibility analysis is used to determine the viability of an idea, such as ensuring a project is legally and [technically feasible](#) as well as economically justifiable. It tells us whether a project is worth the investment—in some cases, a project may not be doable. There can be many reasons for this, including requiring too many resources, which not only prevents those resources from performing other tasks but also may cost more than an organization would earn back by taking on a project that isn't profitable.

A well-designed study should offer a historical background of the business or project, such as a description of the product or service, accounting statements, details of operations and management, marketing research and policies, financial data, legal requirements, and tax obligations. Generally, such studies precede technical development and project implementation.

8.1 Types of Feasibility Study

A feasibility analysis evaluates the project's potential for success; therefore, perceived objectivity is an essential factor in the credibility of the study for potential investors and lending institutions. There are four types of feasibility study—separate areas that a feasibility study examines, described below.

8.1.1 Economical Feasibility

System is economical feasible and can be easily implement with minimum hardware and software resources as this is a cloud based application platform is provided by cloud provider only there is a need of Internet connection and a Browser application. It is very important for designer to first analyze the system economically and determines that project is economical feasible or not. Costs and benefits of the proposed computer system must always be considered together, because they are interrelated and often interdependent. Although the systems analyst is trying to propose a system that fulfills various information requirements, decisions to continue with the proposed system will be based on a cost-benefit analysis, not on information requirements. In many ways, benefits are measured by costs, as becomes apparent in the next section.

Systems analysts are required to predict certain key variables before the proposal is submitted to the client. To some degree, a systems analyst will rely on a what-if analysis, such as, "What if labor costs rise only 5 percent per year for the next three years, rather than 10 percent?" The systems analyst should realize, however, that he or she cannot rely on what-if analysis for everything if the proposal is to be credible, meaningful, and valuable.

The systems analyst has many forecasting models available. The main condition for choosing a model is the availability of historical data. If they are unavailable, the analyst must turn to one of the judgment methods: estimates from the sales force, surveys to estimate customer demand, Delphi studies (a consensus forecast developed independently by a group of experts through a series of iterations), creating scenarios, or drawing historical analogies.

If historical data are available, the next differentiation between classes of techniques involves whether the forecast is conditional or unconditional. Conditional implies that there is an association among variables in the model or that such a causal relationship exists. Common methods in this group include correlation, regression, leading indicators, econometrics, and input/output models.

Unconditional forecasting means the analyst isn't required to find or identify any causal relationships. Consequently, systems analysts find that these methods are low-cost, easy-to-implement alternatives. Included in this group are graphical judgment, moving averages, and analysis of time-series data. Because these methods are simple, reliable, and cost effective, the remainder of the section focuses on them.

- **Estimation of Trends**

Trends can be estimated in a number of different ways. One way to estimate trends is to use a moving average. This method is useful because some seasonal, cyclical, or random patterns may be smoothed, leaving the trend pattern. The principle behind moving averages is to calculate the arithmetic mean of data from a fixed number of periods; a three-month moving average is simply the average of the last three months. For example, the average sales for January, February, and March is used to predict the sales for April. Then the average sales for February, March, and April are used to predict the sales for May, and so on.

When the results are graphed, it is easily noticeable that the widely fluctuating data are smoothed. The moving average method is useful for its smoothing ability, but at the same time it has many disadvantages. Moving averages are more strongly affected by extreme values than by using graphical judgment or estimating using other methods such as least squares. The analyst should learn forecasting well, as it often provides information valuable in justifying the entire project.

- **Tangible Costs**

The concepts of tangible and intangible costs present a conceptual parallel to the tangible and intangible benefits discussed already. Tangible costs are those that can be accurately projected by the systems analyst and the business's accounting personnel.

Included in tangible costs are the cost of equipment such as computers and terminals, the cost of resources, the cost of systems analysts' time, the cost of programmers' time, and other employees
salaries. These costs are usually well established or can be discovered quite easily, and are the costs that will require a cash outlay of the business.

- **Intangible Costs**

Intangible costs are difficult to estimate and may not be known. They include losing a competitive edge, losing the reputation for being first with an innovation or the leader in a field, declining company image due to increased customer dissatisfaction, and ineffective decision making due to untimely or inaccessible information. As you can imagine, it is next to impossible to project a dollar amount for intangible costs accurately. To aid decision makers who want to weigh the proposed system and all its implications, you must include intangible costs even though they are not quantifiable.

8.1.2. Technical Feasibility

It is the study of the function performance and constraints that may affect the ability to achieve an acceptable system. The project development requires designer to have technical knowledge of salesforce.com for both application development and database system. Technical feasibility is one of the most important criteria for selecting material for digitisation. The physical characteristics of source material and the project goals for capturing, presenting and storing the digital surrogates dictate the technical requirements. Libraries must evaluate those requirements for each project and determine whether they can be met with the resources available. If the existing staff, hardware and software resources cannot meet the requirements, then the project will need funding to upgrade equipment or hire an outside conversion agency. If these resources are not available, or if the technology does not exist to meet the requirements, then it is not technically feasible to digitise that material.
Considerations for technical feasibility include:

Image capture: Image capture requires equipment, such as a scanner or a digital camera. Different types of material require different equipment, and different equipment produces images of differing quality. When selecting materials for digitising, technical questions that need to be addressed include: does the original source material require high resolution to capture? Are there any oversized items in the collection? Are there any bound volumes in the

collection? What critical features of the source material must be captured in the digital product? In what condition are the source materials? Will they be damaged by the digitisation process?

Presentation: Presentation refers to how the digitised materials will be displayed online. Consider the following questions to determine the technical feasibility of presenting the digitised material:

Will the materials display well digitally?

How will users use the digital versions?

How will users navigate within and among digital collections?

Do the institutionally supported platforms and networked environment have the capability for accessing the images and delivering them with reasonable speed to the target audience?

Do the images need to be restricted to a specified community?

Do the images need special display features such as zooming, panning and page turning?

Description: Some archival and special collections have been catalogued for public use and contain detailed finding aids with descriptions about each item and the collection as a whole. Other collections may not have been reviewed and documented in detail and do not have much information on individual items. Those collections will require more time, human resources and significant additional expense to research the materials, check the accuracy of the information obtained, and write appropriate descriptions to aid in discovery and use of the digital items. Typewritten documents, like the Drew Pearson columns described above, can have reasonably accurate OCR applied to them to replace, for some uses, the detailed descriptions required for discovery of hand-written or picture materials. The selection criteria should clearly state whether the items and collections that do not contain descriptions should be considered for digitisation.

Human resources: When selecting materials for digitisation, the library should consider whether it has the staff and skill sets to support the digitisation, metadata entry, user interface design, programming and search engine configuration that is required for the project to implement the desired functionality. For large collaborative projects, dedicated staff are usually required from each partner. Digital collections also require long-term maintenance, which needs to be considered and planned for. If a project does not have the necessary staff and skills in-house, but funding is available, outsourcing may be a good choice.

8.1.3. Behavioural Feasibility

In the application domain our system works as an application. There are simple form to fill and service requires no ambiguous entries, all the behavioural entries are simple and GUI based. The system design is very user friendly, interactive. The application should be used by Administrator. People are inherently resistant to change, and computers have been known to facilitate change. An estimate should be made of how strong a reaction the user staff is likely to have toward the development of a computerized system. [t is common knowledge that

computer installations have something to do with turnover, transfers, retraining, and changes in employee job status. Therefore, it is understandable that the introduction of a candidate system requires special effort to educate, sell, and train the staff on new ways of conducting business.

In our safe deposit example, three employees are more than 50 years old and have been with the bank over 14 years, four years of which have been in safe deposit. The remaining two employees are in their early thirties. They joined safe deposit about two years before the study. Based on data gathered from extensive interviews, the younger employees want the programmable aspects of safe deposit (essentially billing) put on a computer. Two of the three older employees have voiced resistance to the idea. Their view is that billing is no problem. The main emphasis is customer service-personal contacts with customers. The decision in this case was to go ahead and pursue the project.

8.1.4   Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.[10]

The operational feasibility assessment focuses on the degree to which the proposed development project fits in with the existing business environment and objectives with regard to development schedule, delivery date, corporate culture and existing business processes.

To ensure success, desired operational outcomes must be imparted during design and development. These include such design-dependent parameters as reliability, maintainability, supportability, usability, producibility, disposability, sustainability, affordability and others. These parameters are required to be considered at the early stages of design if desired operational behaviours are to be realised. A system design and development requires appropriate and timely application of engineering and management efforts to meet the previously mentioned parameters. A system may serve its intended purpose most effectively when its technical and operating characteristics are engineered into the design. Therefore, operational feasibility is a critical aspect of systems engineering that needs to be an integral part of the early design phases

An example of an operational feasibility study, or the fourth type, analyzes the inside operations on how a deemed process will work, be implemented, and how to deal with change resistance and acceptance.

Operational feasibility studies are generally utilized to answer the following questions:

- **Process** – How do the end-users feel about a new process that may be implemented?
- **In-House Strategies** – How will the work environment be affected? How much will it change?
- **Adapt & Review** – Once change resistance is overcome, explain how the new process will be implemented along with a review process to monitor the process change.

If an operational feasibility study must answer the six items above, how is it used in the real world? A good example might be if a company has determined that it needs to totally redesign the workspace environment.

After analyzing the technical, economic, and scheduling feasibility studies, next would come the operational analysis. In order to determine if the redesign of the workspace environment would work, an example of an operational feasibility study would follow this path based on six elements:

- **Process** – Input and analysis from everyone the new redesign will affect along with a data matrix on ideas and suggestions from the original plans.
- **Evaluation** – Determinations from the process suggestions; will the redesign benefit everyone? Who is left behind? Who feels threatened?
- **Implementation** – Identify resources both inside and out that will work on the redesign. How will the redesign construction interfere with current work?
- **Resistance** – What areas and individuals will be most resistant? Develop a change resistance plan.
- **Strategies** – How will the organization deal with the changed workspace environment? Do new processes or structures need to be reviewed or implemented in order for the redesign to be effective?
- **Adapt & Review** – How much time does the organization need to adapt to the new redesign? How will it be reviewed and monitored? What will happen if through a monitoring process, additional changes must be made?

The most important part of operational feasibility study is input—from everyone, especially when it affects how or what an organization does as far as processes. If the process were to build a new sports arena for a client, then a study determining how the arena will operate in a way that is conducive to its inhabitants, parking, human flow, accessibility and other elements is a good example of an operational feasibility study.

Create a sample operational feasibility study if you plan to change something inside the company that will affect how the organization runs or when a client asks you to explore a new product or process that will affect elements within their own organization.

CHAPTER 9

Test Case

9.1 BLACK BOX TESTING :-
The technique of testing without having any knowledge of the interior workings of the application is called blackbox testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a blackbox test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

9.2 WHITE BOX TESTING:-
Whitebox testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called glass testing or openbox testing. In order to perform whitebox testing on an application, a tester needs to know the internal workings of the code.

9.3 GREY BOX TESTING:-
Greybox testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

9.4 UNIT TESTING:-
Unit Testing contains the testing of each unit of Recruitment Application. We have tested each interface by input values and check whether it is working properly working or not we also tested database connectivity. We have entered value in interface and check that the values are properly goes to corresponding tuples or not.

9.5 INTEGRATION TESTING:-
Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottomup integration testing and Topdown integration testing.

9.6 SYSTEM TESTING:-
System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

9.7 ACCEPTANCE TESTING:-
This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement.

CHAPTER 10

CONCLUSION

The package was designed in such a way that future modifications can be done easily.

Automation of the entire system improves the efficiency.

It provides a friendly graphical user interface which proves to be better when compared to the existing system.

It gives appropriate access to the authorized users depending on their permissions.

A software project means a lot of experience. We learned a lot through this project. This project has sharpened our concept game engine, animation, and the software-hardware interface. We learned a lot about different documentation. Now I have much wider knowledge of the features Java offers and put into practice various object-oriented methods that learnt last semester.

CHAPTER 11

LITERATURE REVIEW

Abstract

Microsoft.NET Framework 4 security architecture provides additional capabilities for handling and managing code as part of application security. It provides ways to control the code behavior by enforcing stringent security rules before executing the code. In.NET 4, Code Access Security was simplified. The applications access rights are defined using permissions and their transparency.[1] Code access permission, Identity permissions, and Role-based permission are the various permission types available in.NET 4. Security Transparency separates code into "safe," "unsafe," and "maybe safe" categories. Microsoft.NET 4 Framework advances the capabilities of Allowing Partial Trusted Callers (APTCA) by providing control of permissions to hosts instead of a machinewide policy This chapter demonstrates why we need model-based testing, with a small but complete working example. We exhibit a software defect that is not detected by typical unit tests, but is only exposed by executing more realistic scenarios that resemble actual application program runs [2].

It also looks at building a basic understanding of new additions in.NET Framework 4, such as Dynamic Language Runtime (DLR) and.NET-supported dynamic languages. DLR provides a common framework, a certain set of services, and a runtime execution environment for dynamic languages to function. It provides a pluggable engine to onboard dynamic language compilers through respective binders. DLR is built on top of CLR

11.1 Introduction

C# programs run on .NET, a virtual execution system called the common language runtime (CLR) and a set of class libraries. The CLR is the implementation by Microsoft of the common language infrastructure (CLI), an international standard. The CLI is the basis for creating execution and development environments in which languages and libraries work together seamlessly.[3] Source code written in C# is compiled into an intermediate language (IL) that conforms to the CLI specification. The IL code and resources, such as bitmaps and strings, are stored in an assembly, typically with an extension of *.dll*. An assembly contains a manifest that provides information about the assembly's types, version, and culture.

Similar data can often be handled more efficiently when stored and manipulated as a collection. You can use the System.Array class or the classes in the System.Collections, System.Collections.Generic, System.Collections.Concurrent, and System.Collections.Immutable [4]

C# is an object-oriented programming language [5]. The four basic principles of object-oriented programming are:

- *Abstraction* Modeling the relevant attributes and interactions of entities as classes to define an abstract representation of a system.
- *Encapsulation* Hiding the internal state and functionality of an object and only allowing access through a public set of functions.
- *Inheritance* Ability to create new abstractions based on existing abstractions.
- *Polymorphism* Ability to implement inherited properties or methods in different ways across multiple abstractions.

11.2 Related Work

Software tools are a great aid to process engineers, but too much dependence on such tools can often lead to inappropriate and suboptimal designs. Reliance on software is also a hindrance without a firm understanding of the principles underlying its operation, since users are still responsible for devising the design [6]

Times, when programmers sat in dark cellars and tried to solve all problems on their own are over once and for all. In the meantime software engineering has become a very knowledge-intensive [5] and communicative process (not only but also triggered by agile methods for software development) where the actors heavily exchange data (see Google-Code), connect with like-minded (see Google Summer of Code), blog about experiences in their own weblogs, provide code snippets free of charge (see Django-Snippets) or help

novices with words and deeds in large mailing lists. is social software engineering-the creation of software and related artifacts within a social network-gained a lot of attention in recent software engineering research [7]

11.3 Design(GUI, analysis, implementatiom, evaluation)

The design method that was chosen toward the current Visual C++ demonstration application development consisted of the following steps
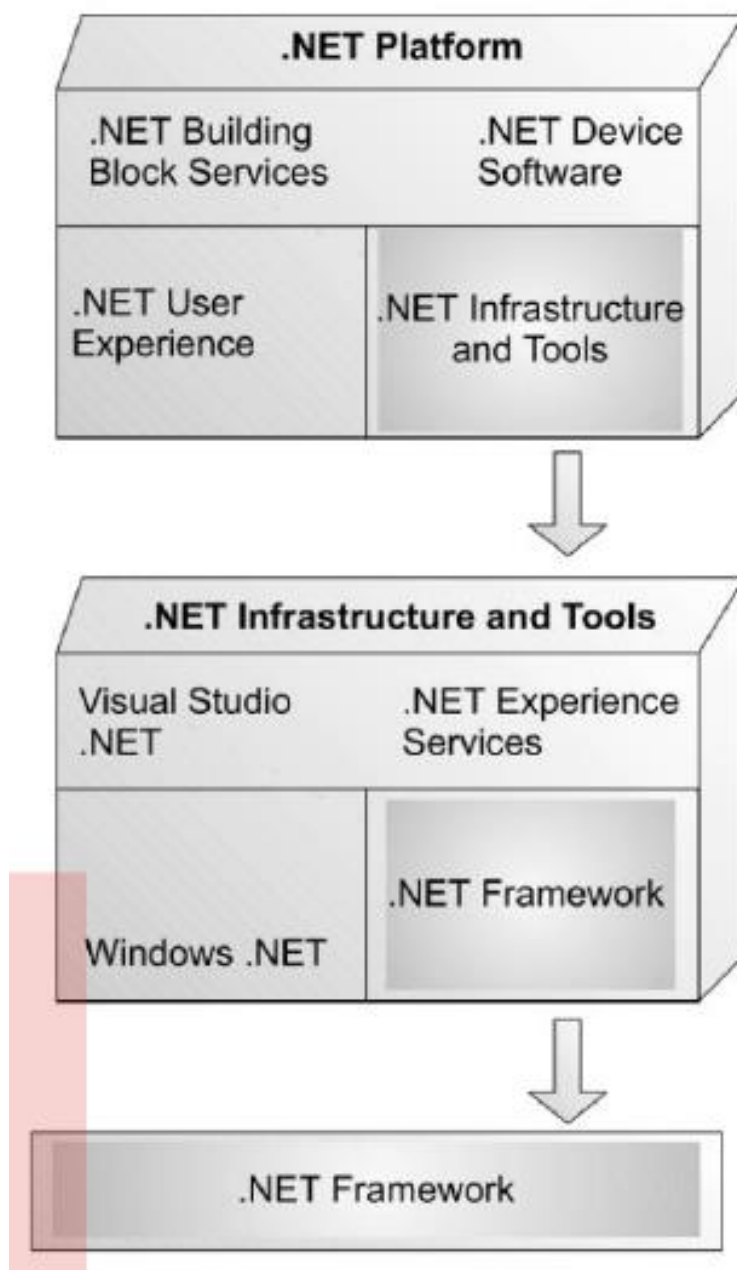
1. Background research on the type of application to be developed 2. Requirements of the intended software application concerning scope and function 3. General use case narratives describing the general workings of the target application 4. Use case scenarios and conversations involving user actions and system responsibilities 5. A noun and verb analysis of the key items and actions concerning the application scope 6. Object analysis involving candidate, responsibilities, and collaborators (CRC) cards 7. Preliminary diagrams reflecting the initial class structure 8. Basic header and source implementing the intentionally bare class structure.
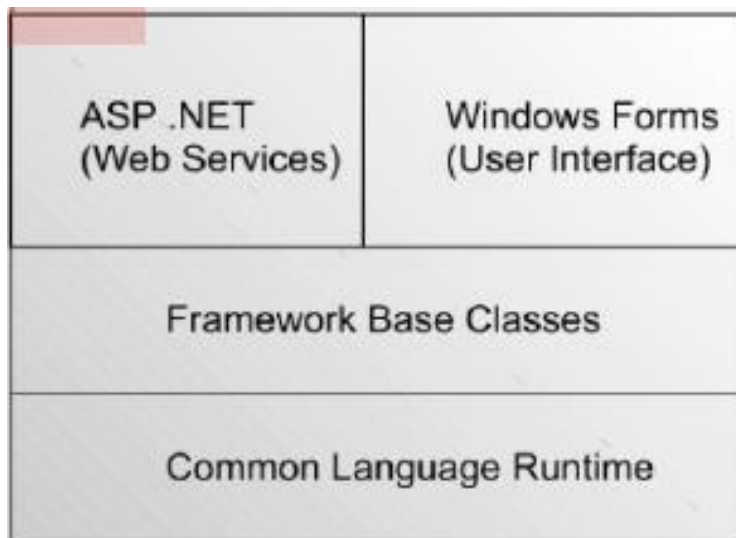
Once the key classes and objects are deigned, a hierarchical class diagram may be drawn to succinctly and visually present the key classes and their association relationships. A preliminary program involving a basic set of source and header Iles organizing and implementing the class structure may then be written, e.g., as a Win32 Console Application, to express the initial set of ideas about how the program should function. This includes basic member methods, variables, and data structures to manage the low of information throughout the program: this is intentionally very brief and concerns only the developer's initial structure, rather than any additional system-provided structure and is designed to be exploratory in nature.[8]

The next step involves the implementation of an initial graphical user interface to display typical Window-based application features, such as child windows, menus with entries, and toolbars with buttons. The Microsoft Visual C++® 6.0, Microsoft Visual Studio Development System (integrated development environment [IDE]), allows the developer to easily set up the interactive features of the application and the key topics covered here are the Application Wizard, menus, icons, toolbars, dialog windows, and attaching functionality to the menu entries and toolbar buttons.

Blocks may be placed on the palette, and connections may be drawn to connect the blocks together. However, before a simulation can be initiated, block parameters need to be set using a dialog window, and the values of these variables need to be updated to the underlying code. Double-left-clicking on any of the blocks should invoke the block dialog window for that particular block and clicking the OK button should update the entered data to the program variables. The following step-by-step instructions indicate how functionality can be added to the code to allow block parameter values to be assigned to a

block via a dialog window invoked upon double-left-clicking a block. Many different types of items may be moved together by its circumscribing them with what is called a "rubber band" and then moving the whole rubber-band-enclosed group together. This is performed using a CRectTracker object and determining whether the enclosed region of the rubber band intersects or contains items on the palette, e.g., blocks, connection bend points, and connection end points, and then updating the positions of the items to be translated by the same amount as the center point of the whole rectangular rubber band region.

```
+----------------------+----------------------+
|   ASP .NET           |   Windows Forms      |
|   (Web Services)     |   (User Interface)   |
+----------------------+----------------------+
|          Framework Base Classes             |
+---------------------------------------------+
|          Common Language Runtime            |
+---------------------------------------------+
```

Going where no book on software measurement and metrics has previously gone, this critique thoroughly examines a number of bad measurement practices, hazardous metrics, and huge gaps and omissions in the software literature that neglect important topics in measurement. The book covers the major gaps and omissions that need to be filled if data about software development is to be useful for comparisons or estimating future projects.

Among the more serious gaps are leaks in reporting about software development efforts that, if not corrected, can distort data and make benchmarks almost useless and possibly even harmful. One of the most common leaks is that of unpaid overtime. Software is a very labor-intensive occupation, and many practitioners work very long hours. However, few companies actually record unpaid overtime. This means that software effort is underreported by around 15%, which is too large a value to ignore. Other sources of leaks include the work of part-time specialists who come and go as needed. There are dozens of these specialists, and their combined effort can top 45% of total software effort on large projects.[9]

The book helps software project managers and developers uncover errors in measurements so they can develop meaningful benchmarks to estimate software development efforts. It examines variations in a number of areas that include:

- Programming languages
- Development methodology
- Software reuse
- Functional and nonfunctional requirements
- Industry type
- Team size and experience

11.4 Results

The programmer-centric book is written in a way that enables even novice practitioners to grasp the development process as a whole.

Incorporating real code fragments and explicit, real-world open-source operating system references (in particular, FreeRTOS) throughout[10], the text:

- Defines the role and purpose of embedded systems, describing their internal structure and interfacing with software development tools
- Examines the inner workings of the GNU compiler collection (GCC)-based software development system or, in other words, toolchain.

The chapter looks at some of the software development paradigms in the world of computer science, including both traditional and advanced methodologies. The traditional software engineering paradigm, based on the classic life cycle, includes the waterfall model of development, incremental development, rapid application development, prototype development (throwaway and exploratory models) and spiral development. Advanced development paradigms include agile development, component-based development, aspect-oriented and cleanroom software development.[11] The major findings and critiques of all these paradigms are discussed in detail.

In particular, this chapter provides an introduction to component-based software engineering, together with the four phases of its evolution, and discusses its characteristics

11.5 Discussion

Starting from a basic definition of models, which refer to simpler mappings of relevant attributes of the real world with the intention to reduce the complexity of the real world with respect to modeling objectives, process models can be understood as a homomorphous, time-based mapping of a real-world system focusing a sequence-based, plausible visualization. According to Krallmann et al. [12], a system to be modeled consists of an amount of system elements, that are connected with an amount of system relations. As it is limited by a system border, the system environment and the system are connected with an interface to exchange system input and system output.

Failure of safety critical applications might lead to serious consequences such as significant financial loss or even loss of life.[13] Thus, software quality assurance, also simply called software assurance, has become the focus when certifying a safety critical system. Software assurance includes reliability, security, robustness, safety, and other quality-related attributes, as well as functionality and performance.

11.6 Conclusion

Knowledge production within the field of business research is accelerating at a tremendous speed while at the same time remaining fragmented and interdisciplinary. This makes it hard to keep up with state-of-the-art and to be at the forefront of research, as well as to assess the collective evidence in a particular area of business research. This is why the literature review as a research method is more relevant than ever. Traditional literature reviews often lack thoroughness and rigor and are conducted ad hoc, rather than following a specific methodology. Therefore, questions can be raised about the quality and trustworthiness of these types of reviews.[14] This paper discusses literature review as a methodology for conducting research and offers an overview of different types of reviews, as well as some guidelines to how to both conduct and evaluate a literature review paper. It also discusses common pitfalls and how to get literature reviews published.

the process of digitalisation, and accepted standards in these fields are essential for building and exploiting complex computing, communication, multimedia and measuring systems.[15] Standards can simplify the design and construction of individual hardware and software components and help to ensure satisfactory interworking.

11.7 References

[1] .NET 4 for Enterprise Architects and Developers

[2] Model-Based Software Testing and Analysis with C#

[3] PROGRAMMING IN C#

[4] Data Structures and Algorithms Using Visual Basic.NET

[5] Object-Oriented Programming with Visual Basic.NET

[6] Process Engineering and Design Using Visual Basic

[7] Data Structure and Software Engineering

[8] Software Application Development

[9] A Guide to Selecting Software Measures and Metrics

[10] Embedded Software Development

[11] Component-Based Software Engineering

[12] Business Modeling and Software Design

[13] Dependable Software Engineering. Theories, Tools, and Applications

[14] Computer Science Review

[15] Computer Standard & Interfaces