SSE 657

Object-Oriented Project Methods

Project #3

by

Jason Payne

December 5, 2014

TABLE OF CONTENTS

Topics Covered	Topic Examples
Iterative Development	• OOAD Software Lifecycle • Feature Driven Development • Use-Case Driven Development

1. The *'Recipe Helper'* Application

For the sake of this project, a software application will be developed according to the Object Oriented Analysis & Design (OOAD) paradigm. This project will demonstrate OOAD concepts by applying the concepts at different stages of development of an application (Figure 1-1). Each section (highlighted in red) will focus on specific phases of the OOAD project lifecycle as it applies to the current development stage of the application.

Two iterations of the software development cycle will be presented as topic examples of OOAD concepts. The first iteration will be driven by a common use-case scenario (Use-Case Driven Development) that exercises the application from beginning to end. The next iteration will focus on the implementation of a main application feature (Feature Driven Development). While not all features will be implemented in this project, it is still important to include them in the early stages of development for the purposes of defining requirements and determining risks. Once past those early stages, the project will shift focus to the specific development iterations.

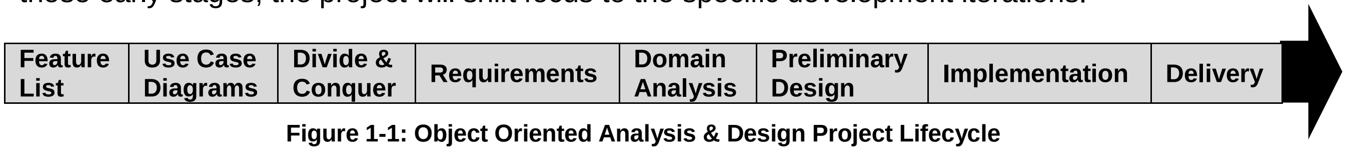


Figure 1-1: Object Oriented Analysis & Design Project Lifecycle

1.1 Vision Statement

The name of the application is simply *'Recipe Helper'* (henceforth referred to as the "application") and is intended for use by restaurant entrepreneurs and private chefs. This application will provide a simple mechanism for maintaining and organizing a collection of recipes while also helping create an itemized list of items required by the recipes.

Vision Statement for the 'Recipe Helper'
This application will provide chefs – private and commercial alike – with a convenient tool for organizing & maintaining recipes while also creating an itemized list of ingredients required by the recipe. Each recipe should be grouped according to the time of day it is served (e.g. breakfast, sides, entree, dessert) so that groups of recipes can be bundled together for the purposes of creating a menu. Once a menu is created, the application can then be used to generate a grocery list composed of the necessary ingredients required by the recipes.

2. Feature List

No matter what development approach a developer decides to take, the first few steps are always the same because it is incredibly difficult to start writing any code unless the customer's needs and desires are understood first. With that said the first thing to resolve is the issue of determining the main features that the application must provide.

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

2.1 Customer Conversation

When determining the main features, the most obvious place to start is with communications with the customer(s). By setting up a dialog early in the development lifecycle, it establishes good lines of communication and trust between developer and customer. It also gives the customer confidence that you are interested in their needs and desires. Below is an example conversation that could take place between customer and developer after the developer has taken some time to analyze the vision statement and expose some "hidden" requirements based on deduction.

Developer's questions about vision statement:

1. Can recipes exist in multiple meal groups? It is plausible – for example – that a recipe could be a side dish and also be considered a main dish.

   Customer's response: *Good point! Yes, recipes should be able to be in multiple meal groups. It will be up to each user to ensure the accuracy of how their recipes are grouped.*

2. Should customizable meal groups be supported by this application?

   Customer's response: *This is a good idea for a future version, but not necessary for this version of the application.*

3. Should multiple menus (i.e. one for breakfast, one for lunch, one for dinner) be maintained by the application or should the application create only a single menu?

   Customer's response: *This is probably a good idea for a future version, but not necessary for this version of the application. However, the menu should be able to be saved to a text file.*

4. Should the menu(s) have a name, such as the name of the restaurant or something similar?

   Customer's response: *Yes! Good idea.*

5. In the case that a single recipe could be consumed multiple times between inventory restock, should the grocery list be created using a multiplication factor based on the estimated amount of times that a single recipe will be consumed?

   Customer's response: *Great point and obviously yes! It makes sense to need to multiply the ingredients by the number of possible times that a recipe could be ordered.*

6. How should the grocery list be organized?

   Customer's response: *The grocery list should be listed in alphabetical order.*

After analysis and discussion with the customer is complete, a feature list can be generated. The feature list for this application is presented below.

Table 2-1: *'Recipe Helper'* Feature List

Feature ID	Feature (from customer / vision statement)
1	The app shall maintain a collection of recipes categorized by meal groups.
2	The app shall support the creation of named menus from a group of recipes.
3	The app shall support saving menus to text files.
4	The app shall support the creation of grocery lists based upon the recipes from a menu.

3. Use Case Diagram

The next step is to determine what the use cases will be for the application. The focus should still remain at the feature level, so any thoughts focused on lower-level details should be spared for later stages in the development lifecycle. The best way to keep the focus at the feature level is with a use-case diagram (Figure 3-1).

Feature List	**Use Case Diagrams**	Divide & Conquer	Requirements	Domain Analysis	Preliminary Design	Implementation	Delivery

While it is not necessary that every use case “bubble” relate to a feature, it is necessary that every feature be covered by a “bubble” (as seen below). The detailed use-cases are presented in the subsequent sections below.

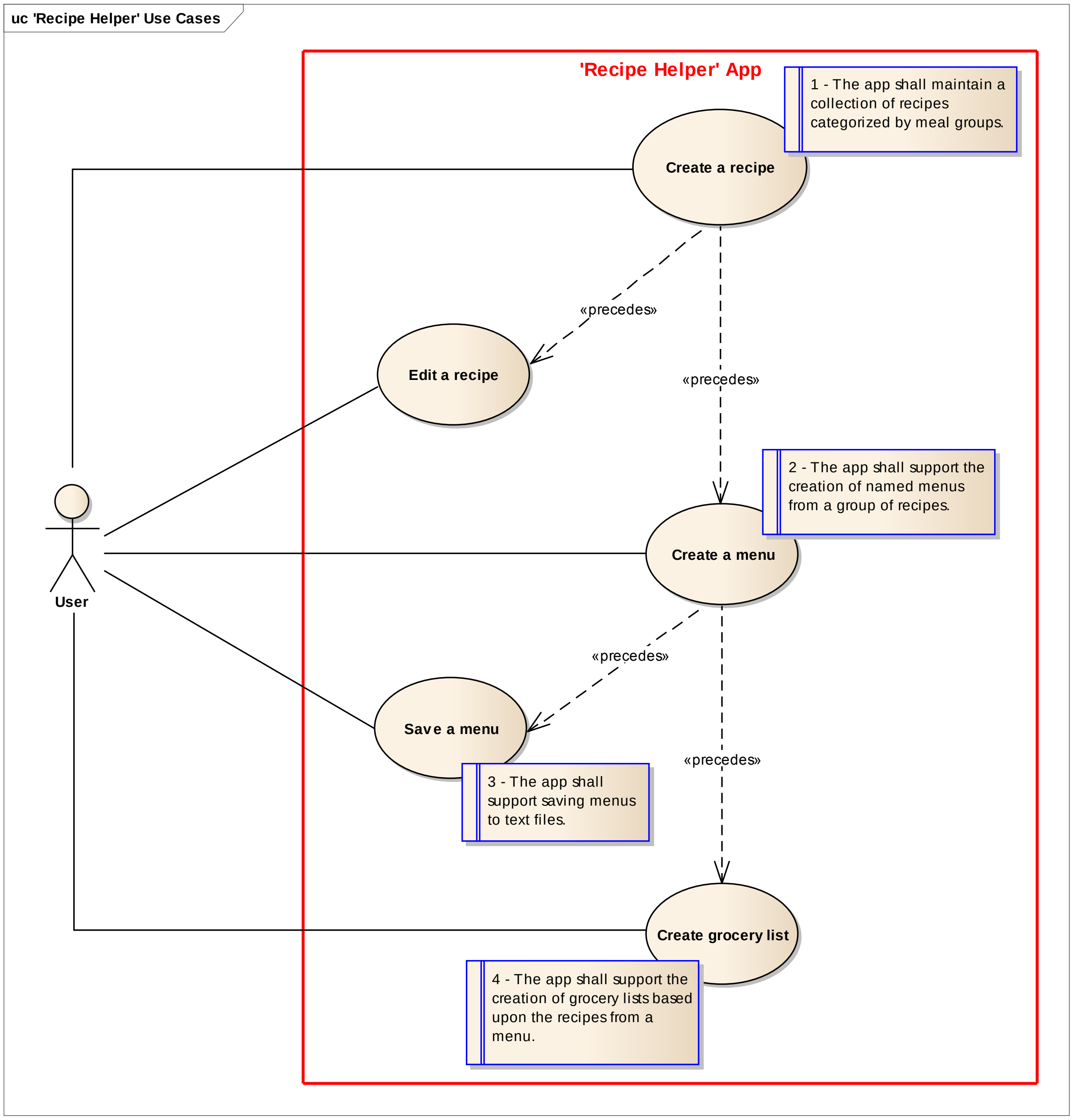

uc 'Recipe Helper' Use Cases
'Recipe Helper' App
1 - The app shall maintain a collection of recipes categorized by meal groups.
Create a recipe
«precedes»
Edit a recipe
«precedes»
2 - The app shall support the creation of named menus from a group of recipes.
User
Create a menu
«precedes»
Save a menu
3 - The app shall support saving menus to text files.
«precedes»
Create grocery list
4 - The app shall support the creation of grocery lists based upon the recipes from a menu.

Figure 3-1: Application Use Case Diagram

3.1 Use Case: Create a Recipe

Use Case: Create a recipe
1. From the main form, the user selects the 'Add Recipe…' button. 2. From the 'Edit Recipe' form, the user enters the name, ingredients, and cooking steps of the recipe. 2.1 Optionally, the user enters the prep time and/or cooking time (in hours/minutes such as "HH:MM") of the recipe. 3. The user selects the meal group type(s) of the recipe. 4. Once steps 2 – 5 have been completed, the user selects the 'Save' button to store the recipe to the collection of recipes. 4.1 The user selects the 'Cancel' button to disregard the changes without saving.

3.2 Use Case: Edit a Recipe

Use Case: Edit a recipe
1. From the main form, the user selects a recipe from the list of recipes and selects the 'Edit Recipe…' button. 2. From the 'Edit Recipe' form, the user edits the name, ingredients, cooking steps, prep time, cooking time, and/or meal group type(s) of the recipe. 3. The user selects the 'Save' button to save the changes to the recipe. 3.1 The user selects the 'Cancel' button to disregard the changes without saving.

3.3 Use Case: Create a Menu

Use Case: Create a menu
1. From the main form, the user selects any number of recipes from the list of recipes. 2. The user selects the 'Create Menu…' button. 3. A preview of the menu is presented to the user with options to close the menu preview ('Close'), save the menu to a text file (see **Use Case: Save a menu**), or print a grocery list from the menu (see Create grocery list).

3.4 Use Case: Save a Menu

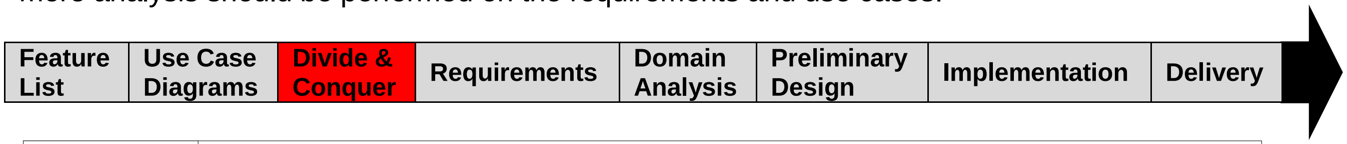
Use Case: Save a menu
1. The user creates a menu (see Use Case: Create a menu). 2. From the menu preview, the user selects the 'Save Menu…' button. 3. The user is prompted to save the menu to a text file. 4. The user selects the 'Save' button to save the menu to a text file at the desired location. 4.1 The user selects the 'Cancel' button to exit the prompt without saving and return to the menu preview form.

3.5 Use Case: Create Grocery List

Use Case: Create grocery list
1. The user creates a menu (see Use Case: Create a menu). 2. From the 'Grocery List' section of the 'Menu Preview' form, the user goes through the list of recipes and specifies a multiplication factor for each recipe that represents the estimated number of times the recipe will be prepared. The default is 1. 3. The user selects the 'Print Grocery List…' button to print the grocery list. 4. From the print dialog, the user selects their printer and prints the grocery list as needed.

4. Divide & Conquer

After the requirements have been established and some use-case scenarios have been identified, a high-level view of the system should be able to be formulated. Each section of the system should be considered a module that can be isolated as a single piece of the system. If there is still trouble in separating the application into modules, either the scope is too small or more analysis should be performed on the requirements and use cases.



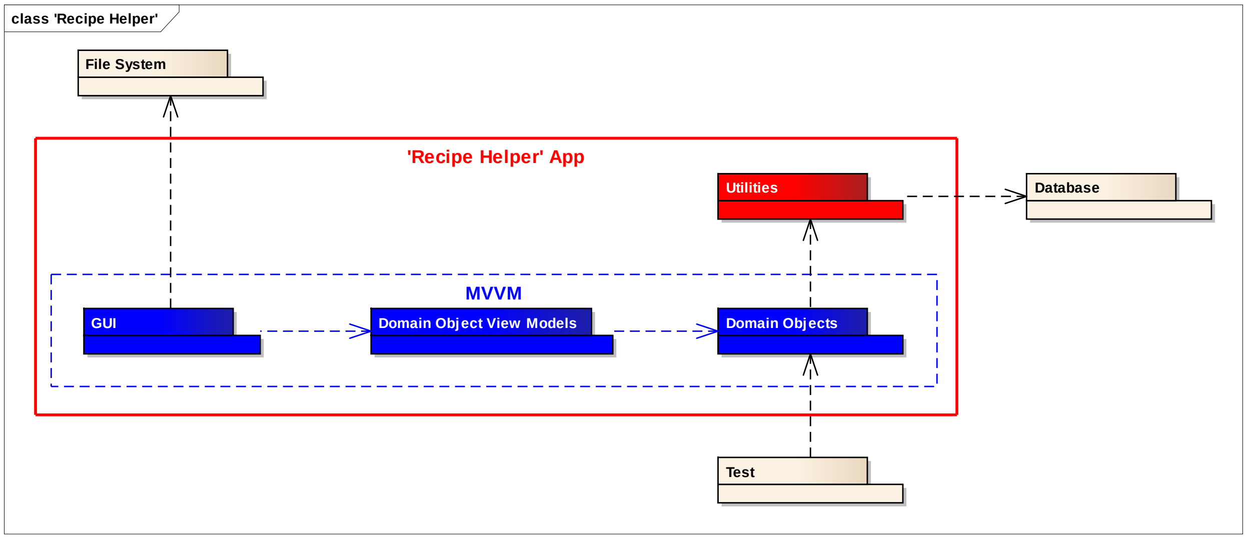


Figure 4-1: High-Level System Module Diagram

Blue packages = MVVM objects within the system boundary
Red packages = non-MVVM objects within the system boundary

From Figure 4-1, the modules of the system can be seen with respect to the application's logical boundaries. It can also be seen that the MVVM design pattern can be utilized when creating the WPF-based GUI. A few external modules have been identified that fall outside of the scope of the system. These can sometimes be external systems that the developer has no control over and/or frameworks that will be used by application. In this diagram, it can be seen that three external modules exist: one for test objects (unit tests), one for database services used for data persistence, and one for the file system (saving, printing, etc.).

4.1 Risk Analysis

The next step is to identify the architectural characteristic of each feature. This is accomplished by asking the following questions as they relate to each feature:

- **Essence** – is the component a core part of the system? Can the system exist with that component? If not, then that component is a core part of the system.
- **Meaning** – is there a lack of clarity or meaning for the component? Uncertainty of a component could take a good deal of time to gain clarity. Therefore, it is better to spend time on such components earlier rather than later.
- **Unfamiliarity** – how can a component or feature be implemented? If it is a difficult task or an unfamiliar task to get implemented, it is better to spend time on such components earlier rather than later.

4.1.1 <u>Architecture</u>
Table 4-1 illustrates how the "3 Q's of Architecture" have been applied to the application and its feature set.

Table 4-1: Architectural Characteristics of the 'Recipe Helper' Feature Set

Feature ID	Feature (from customer / vision statement)	Architecture Characteristic
1	The app shall maintain a collection of recipes categorized by meal groups.	- Essence - Unfamiliar
2	The app shall support the creation of named menus from a group of recipes.	- Essence
3	The app shall support saving menus to text files.	- Essence
4	The app shall support the creation of grocery lists based upon the recipes from a menu.	- Essence - Unfamiliar

4.1.2 <u>Risks</u>
Now that the architecture analysis has been completed for the feature set, a prioritized list – based on greatest risk – can be generated in order to help determine where development should begin. From the table above, it can be seen that features 1 and 4 appear to provide the greatest risk, so that is where the first iterations of development should focus. Now, deciding the singular greatest risk requires more detailed analysis, which reveals:

1) According to feature 1, a collection of data will be maintained by the application. To maintain data persistence between sessions, storing recipes in a database would provide a solid solution. However, database setup and design can be time-consuming and since a collection of recipes can be mimicked (or mocked) at run-time, it may be possible to push this off if another feature is more essential to providing a working demo to the customer(s).
2) Feature 4 appears to be a common scenario based upon the vision statement and further discussions with the customer. Given that the behavior of feature 1 can be mimicked with mock objects, feature 4 should be the starting point as it would provide the customer(s) with a good demonstration product and would also provide a good baseline for future development iterations to work from.

5. Iterative Development: Use-Case Driven Development

For the first development iteration, a use-case driven approach was chosen due to the fact that by focusing on a common focus scenario, a good foundation could be established for future development iterations. Several hurdles were able to be addressed in this iteration such as GUI implementation with the MVVM pattern, domain object definitions, and public interface definitions. Every detail and use-case was not addressed in this iteration, but that was not the goal. The goal was to focus on providing enough implementation such that a demonstration could be provided to the customer(s) (which is shown at the end of this section).

5.1 Requirements

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

At this point, the analysis has been completed and a feature list with use-cases has been established. Using that information, a primary use-case will serve as the driving requirements for this development iteration. That primary use-case, or focus scenario, is presented below.

Focus Scenario: Creating a Grocery List

1. From the main form, the user selects any number of recipes from the list of recipes.
2. The user selects the 'Create Menu…' button.
3. A preview of the menu is presented to the user.
4. From the 'Grocery List' section of the 'Menu Preview' form, the user goes through the list of recipes and specifies a multiplication factor for each recipe that represents the estimated number of times the recipe will be prepared. The default is 1.
5. The user selects the 'Print Grocery List…' button.
6. The user is presented with the standard print dialog.
7. The user selects the 'Print…' button to print the grocery list.

🟦 = Candidate classes
🟩 = Candidate methods/attributes

5.2 Domain Analysis

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

Using the focus scenario, a list of candidate classes, methods and attributes are identified to help reduce delays in development by keeping the focus strictly on what is needed now. Those results are shown in the tables below.

Table 5-1: Iteration #1 Candidate Classes

Candidate Classes	Developer Notes
Main form	▪ Main GUI window (WPF) ▪ needs 'Create Menu…' button
Recipe	▪ Critical domain object ▪ needs attributes for name, ingredients, cooking steps, prep time, cooking time
List of Recipes	▪ Can be represented by a standard .NET collection
~~'Create Menu…' button~~	▪ Included as part of the main form

Candidate Classes	Developer Notes
Menu preview	▪ WPF Dialog presented to the user as a preview of a menu. ▪ needs 'Grocery List' section for the creation of grocery lists ▪ needs 'Print Grocery List…' button that initiates printing services
~~'Grocery List' section~~	▪ Included as part of the Menu preview form
Multiplication factor	▪ number of times the recipe will be prepared
~~'Print Grocery List…' button~~	▪ Included as part of the Menu Preview form
~~Standard print dialog~~	▪ standard Windows print dialog
Ingredient	▪ singular form of a "grocery" object
Grocery List	▪ Can be represented by a standard .NET collection
~~'Print…' button~~	▪ Included as part of the 'Grocery List' form

Table 5-2: Iteration #1 Candidate Methods/Attributes

Candidate Methods/Attributes	Developer Notes
~~Select~~	▪ By context, will be handled by .NET/UI
~~Presented~~	▪ By context, will be handled by .NET/UI
~~Specifies~~	▪ By context, will be handled by .NET/UI
Print Grocery List	▪ Initiates the service for printing grocery lists ▪ UI event handler

5.3 Preliminary Design

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

With the focus scenario serving as the driving requirements and a list of candidate objects, a preliminary design can be created (Figure 5-1).

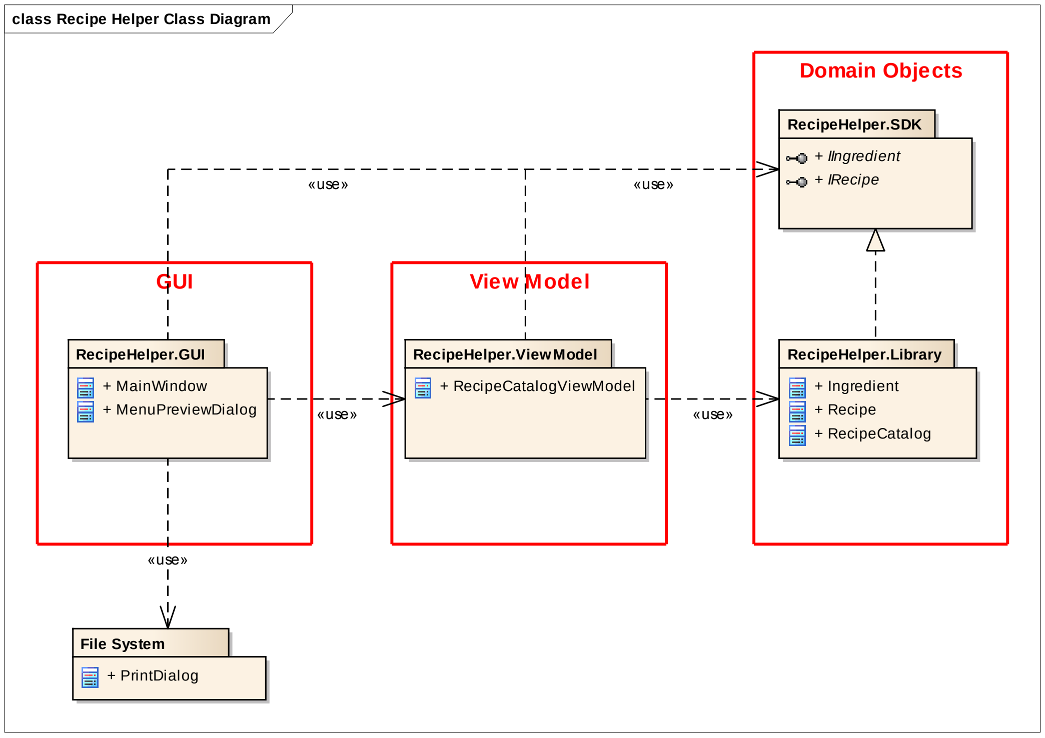


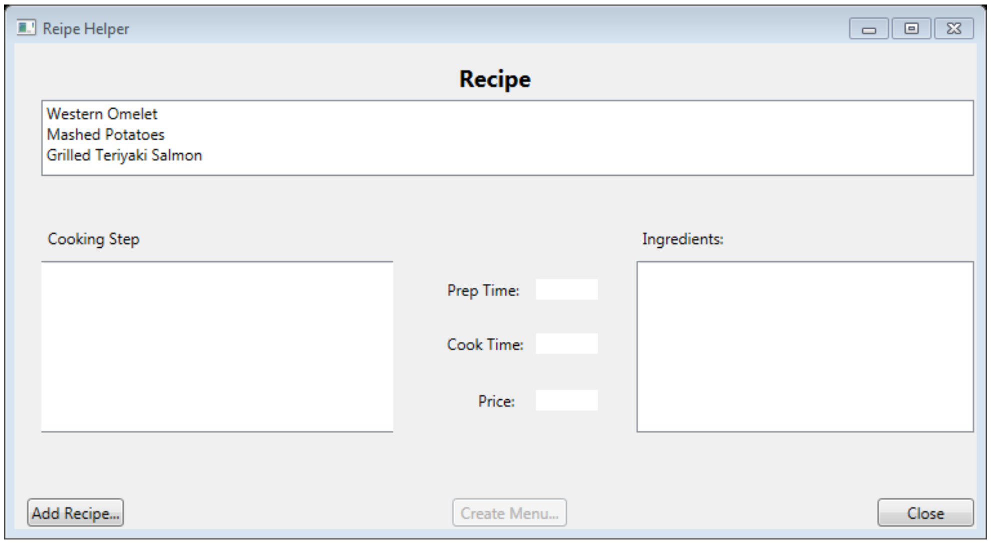
Figure 5-1: Class Diagram for Iteration #1

5.4 Implementation

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

The source code can be found in Appendix A – *'Recipe Helper' Source Code*, however, the following steps will serve as insight into how the code was developed as each step matches with a step from the focus scenario.

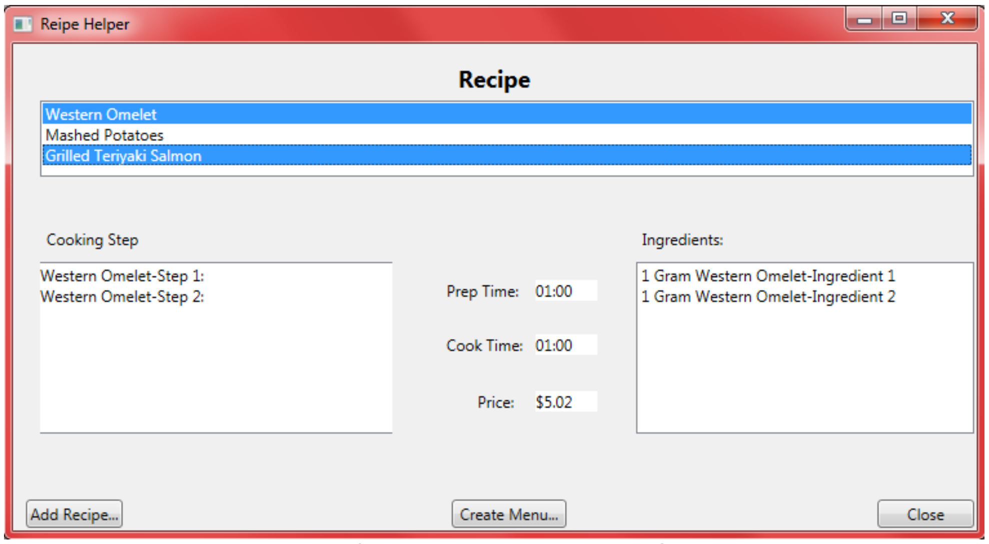
<u>Step 1:</u> Present the main form and allow the user to select any number of recipes from the recipes list.



Western Omelet
Mashed Potatoes
Grilled Teriyaki Salmon

Figure 5-2: Main window presented at run-time

<u>Step 2:</u> After making selections, the user selects the 'Create Menu…' button.



Western Omelet
Mashed Potatoes
Grilled Teriyaki Salmon

Western Omelet-Step 1:
Western Omelet-Step 2:

Prep Time: 01:00
Cook Time: 01:00
Price: $5.02

1 Gram Western Omelet-Ingredient 1
1 Gram Western Omelet-Ingredient 2

Figure 5-3: Main window after recipes selected (and 'Create Menu…' enabled)

Step 3: Present a preview of the menu to the user.

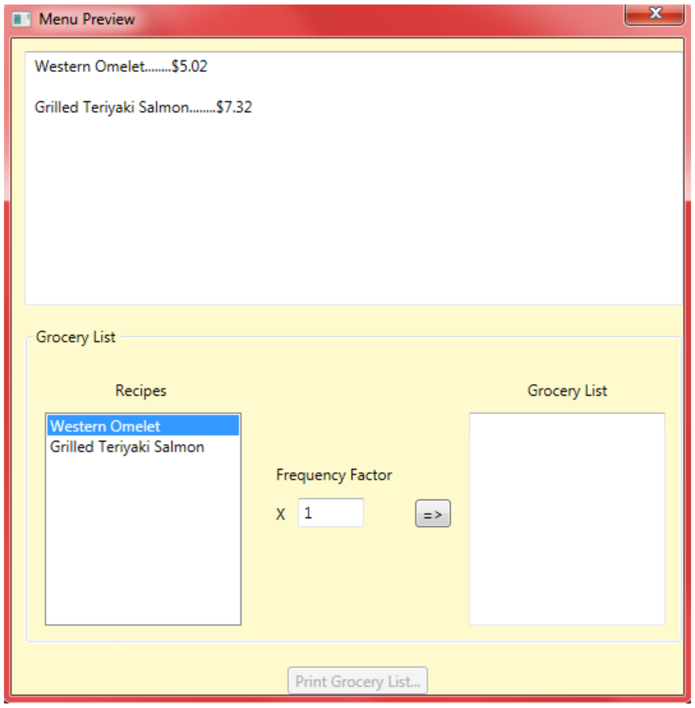


Figure 5-4: Menu Preview window (with 'Print Grocery List…' button disabled)

Step 4: The user goes through the list of recipes applying a value (Frequency Factor) for the number of times a recipe is expected to be prepared.

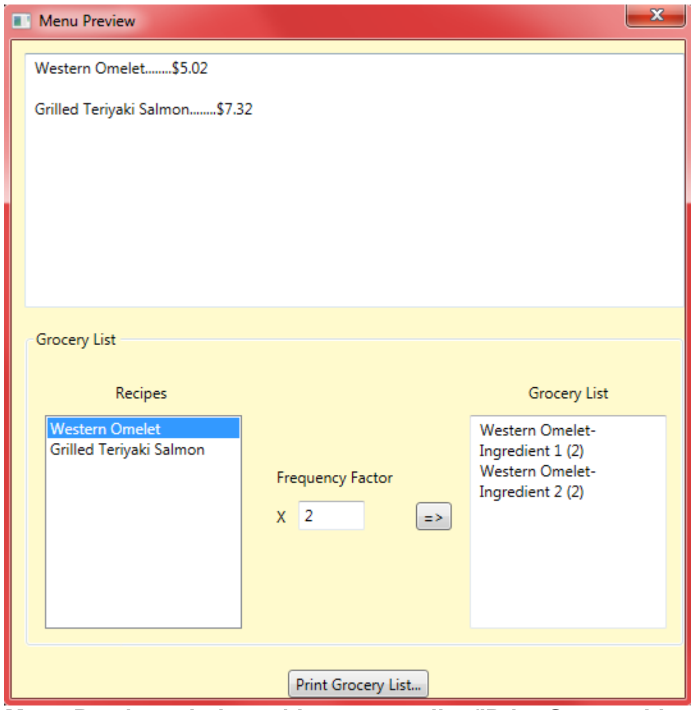
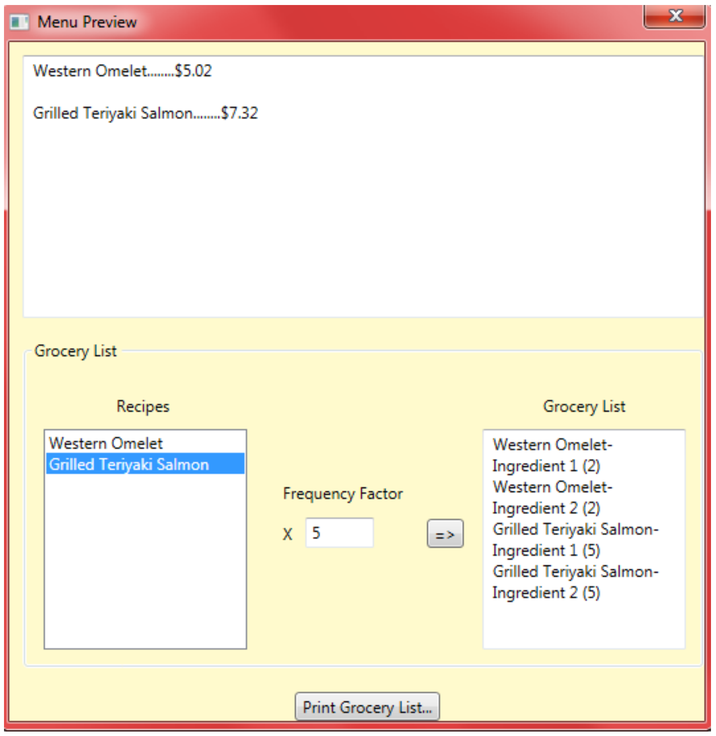

Menu Preview
Western Omelet........$5.02
Grilled Teriyaki Salmon........$7.32
Grocery List
Recipes
Grocery List
Western Omelet
Grilled Teriyaki Salmon
Frequency Factor
X 2
=>
Western Omelet-Ingredient 1 (2)
Western Omelet-Ingredient 2 (2)
Print Grocery List...

Figure 5-5: Menu Preview window with a grocery list ('Print Grocery List…' enabled)


Menu Preview
Western Omelet........$5.02
Grilled Teriyaki Salmon........$7.32
Grocery List
Recipes
Western Omelet
Grilled Teriyaki Salmon
Frequency Factor
X 5
=>
Grocery List
Western Omelet-Ingredient 1 (2)
Western Omelet-Ingredient 2 (2)
Grilled Teriyaki Salmon-Ingredient 1 (5)
Grilled Teriyaki Salmon-Ingredient 2 (5)
Print Grocery List...
Figure 5-6: Example of how the Frequency Factory updates the grocery list

<u>Step 5:</u> The user selects the 'Print Grocery List…' button and is presented with the standard print dialog.

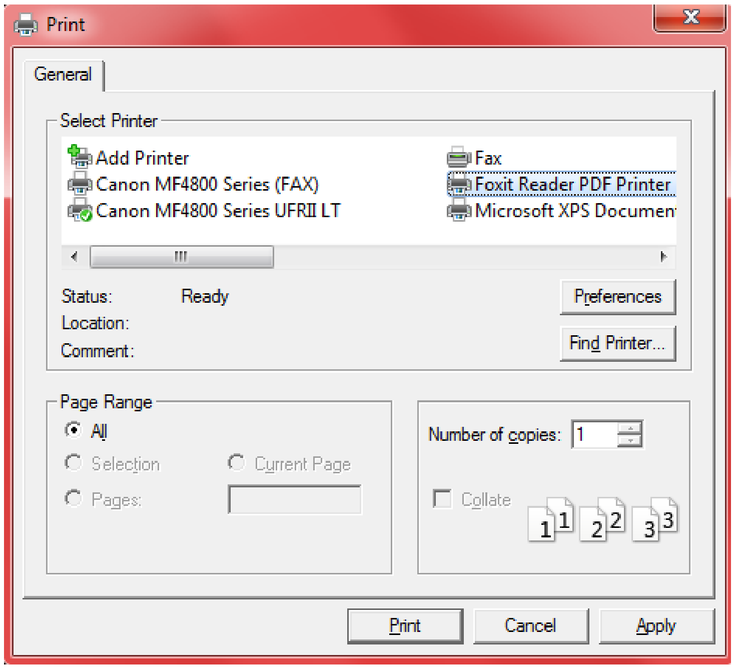
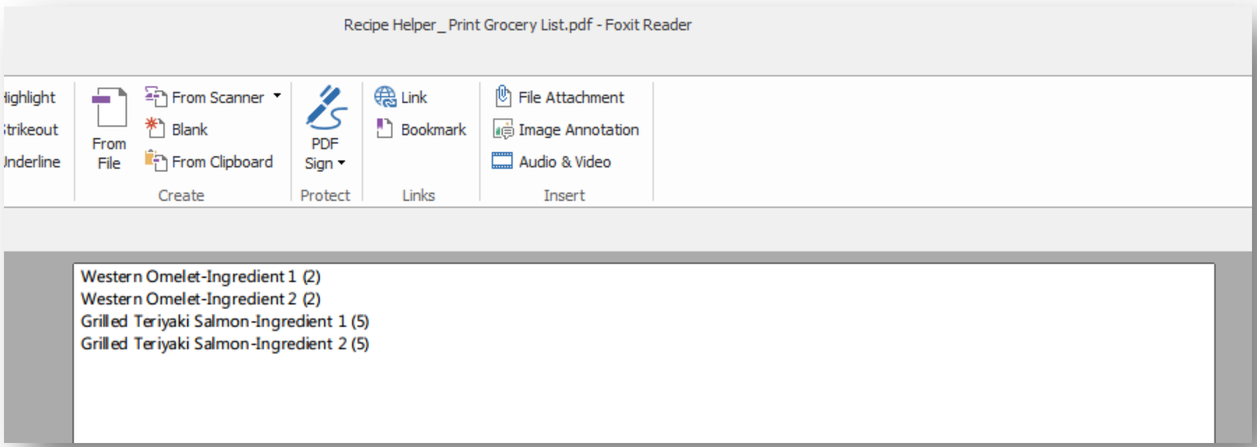


Figure 5-7: Standard printing dialog provided by the application

<u>Step 6:</u> The user selects the 'Print…' button to print the grocery list.



Western Omelet-Ingredient 1 (2)
Western Omelet-Ingredient 2 (2)
Grilled Teriyaki Salmon-Ingredient 1 (5)
Grilled Teriyaki Salmon-Ingredient 2 (5)

Figure 5-8: Screen capture of printed grocery list

6. Iterative Development: Feature Driven Development

For this development iteration, a feature driven approach was chosen because with the extensive work done in iteration #1, one of the main application features could be added efficiently. This iteration focuses on implementing feature 3 which involves saving menus to a text file. Since a lot of the foundation for the code base was established in iteration #1, the general thought is that this should involve minor impact to design and implementation. But this is expected because a lot of the work completed in iteration #1 applies to all subsequent iterations!

6.1 Requirements

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

As before, a focus scenario will be used to drive requirements, but this use case is centered on the main application feature related to saving menus to text files. That focus scenario is presented below.

Focus Scenario: Saving a Menu To File
1. From the main form, the user selects any number of recipes from the list of recipes.
2. The user selects the 'Create Menu…' button.
3. A preview of the menu is presented to the user.
4. The user selects the 'Save Menu…' button.
5. The user is presented with the standard save file dialog and specifies a location and filename where the menu file should be saved.

🟦 = Candidate classes
🟩 = Candidate methods/attributes

6.2 Domain Analysis

Feature List	**Use Case Diagrams**	**Divide & Conquer**	**Requirements**	**Domain Analysis**	**Preliminary Design**	**Implementation**	**Delivery**

Using the focus scenario, a list of candidate classes, methods, and attributes are identified. There is a lot of similarity, but again, this is not unexpected due to the efforts in development iteration #1.

Table 6-1: Iteration #2 Candidate Classes

Candidate Classes	Developer Notes
~~Main form~~	▪ Completed in implementation iteration #1
~~Recipe~~	▪ Completed in implementation iteration #1
~~List of Recipes~~	▪ Completed in implementation iteration #1
~~'Create Menu…' button~~	▪ Completed in implementation iteration #1
Menu preview	▪ Needs 'Save…' button
~~'Save…' button~~	▪ Included as part of the Menu Preview form
~~Save file dialog~~	▪ Standard Windows save file dialog

Table 6-2: Iteration #2 Candidate Methods/Attributes

Candidate Methods/Attributes	Developer Notes
~~Select~~	▪ By context, will be handled by .NET/UI
~~Presented~~	▪ By context, will be handled by .NET/UI

Candidate Methods/Attributes	Developer Notes
~~Specifies~~	▪ By context, will be handled by .NET/UI
Save	▪ Initiates the service for saving menus ▪ UI event handler

6.3 Preliminary Design

Feature List	Use Case Diagrams	Divide & Conquer	Requirements	Domain Analysis	Preliminary Design	Implementation	Delivery

The design impact is minimal as a dependency on the SaveDialog is required, but all else remains the same.

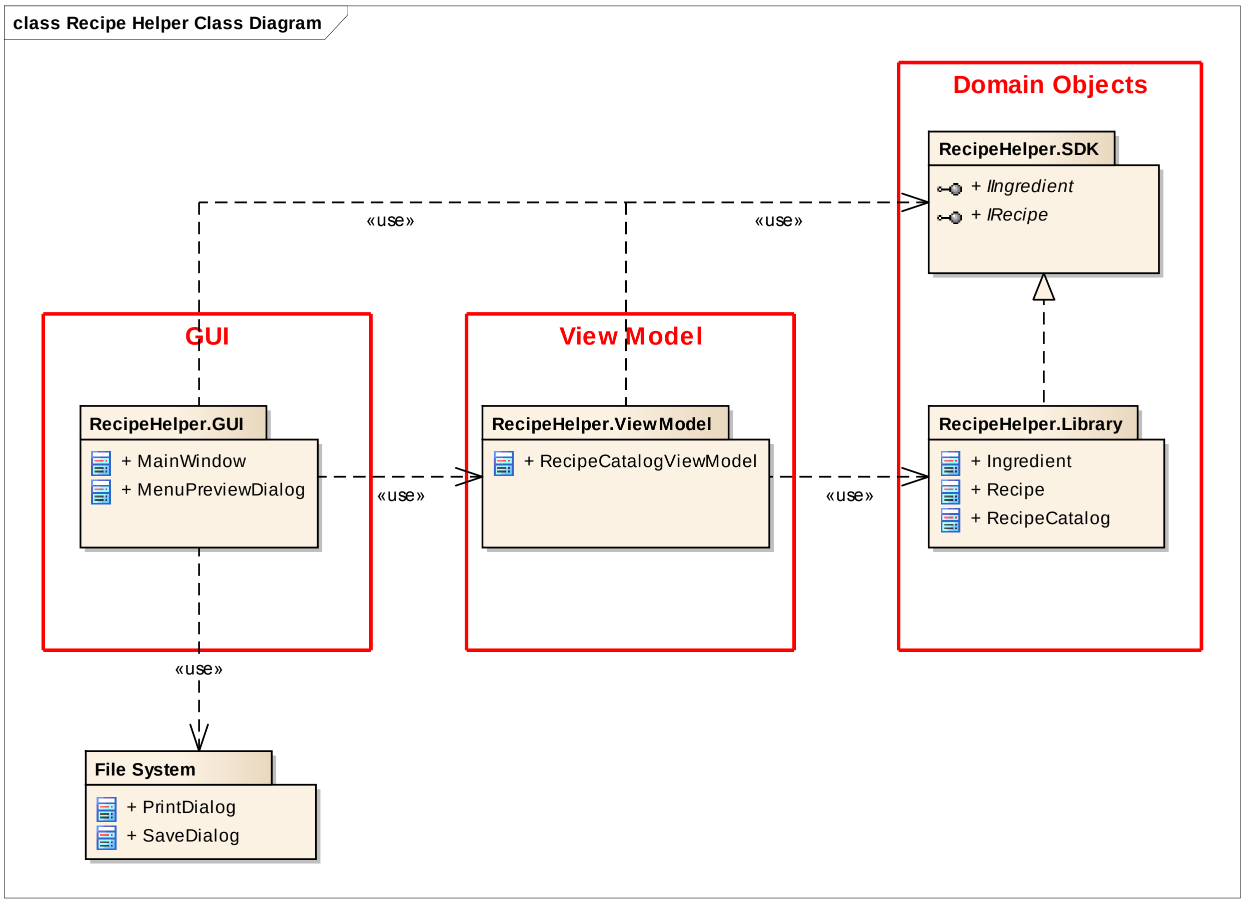


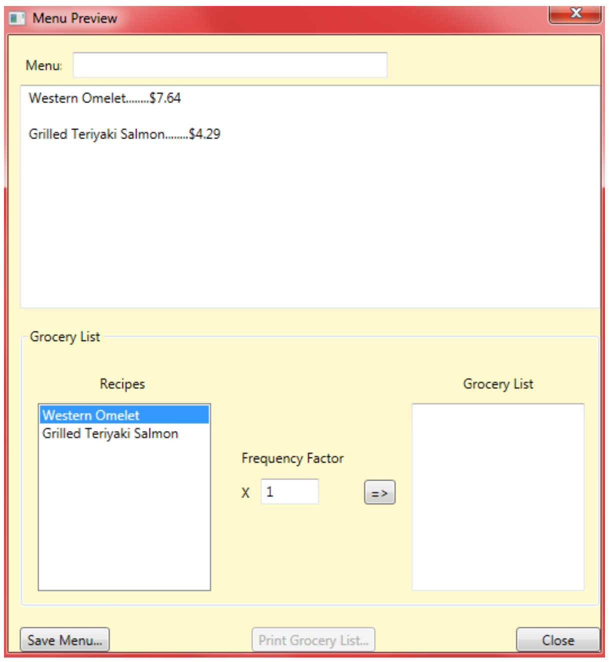
Figure 6-1: Class Diagram for Iteration #2

6.4 Implementation

Feature List	Use Case Diagrams	Divide & Conquer	Requirements	Domain Analysis	Preliminary Design	Implementation	Delivery

The source code can be found in Appendix A – *'Recipe Helper' Source Code*, however, the following steps will serve as insight into how the code was developed as each step matches with a step from the focus scenario.

Step 1: User selects any number of recipes from the list of recipes (Figure 5-3).
Step 2: User selects the 'Create Menu…' button and is presented with the Menu Preview form.



Western Omelet........$7.64

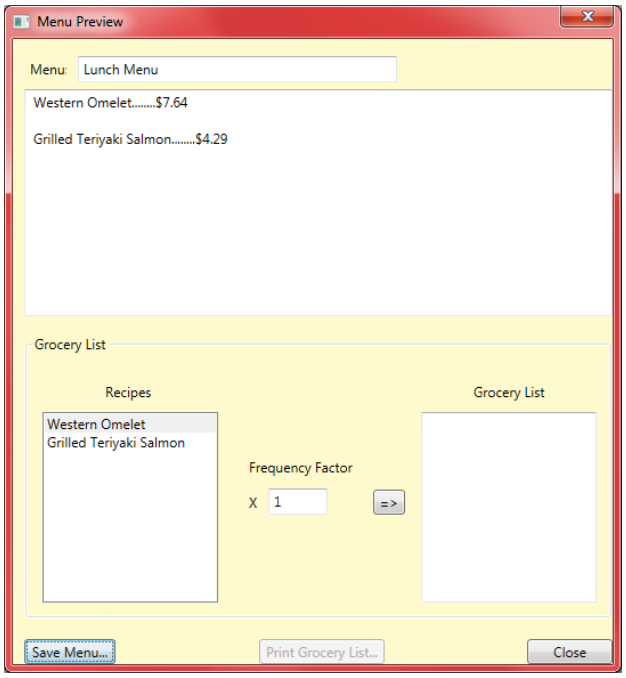
Grilled Teriyaki Salmon........$4.29

Western Omelet
Grilled Teriyaki Salmon

X 1

Figure 6-2:Menu Preview window (now including a menu name editor and a 'Save Menu…' button)

<u>Step 3:</u> User gives the menu a name and selects the 'Save Menu…' button.



Menu: Lunch Menu

Western Omelet........$7.64

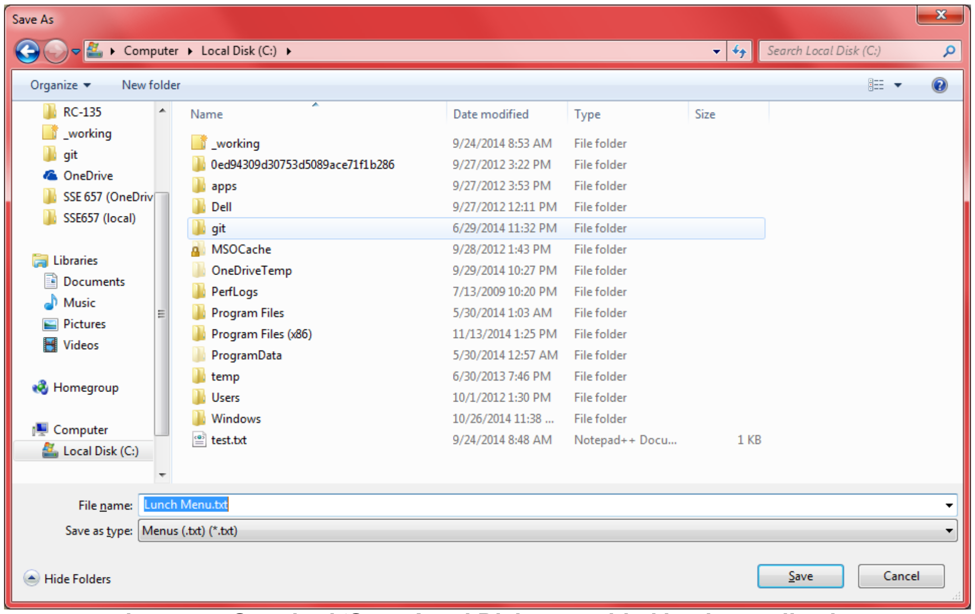
Grilled Teriyaki Salmon........$4.29

Western Omelet
Grilled Teriyaki Salmon

X 1

Figure 6-3:Menu Preview window (with a menu name specified)

<u>Step 4:</u> The 'Save As…' dialog is presented to the user with the default file name based off of the menu name. User selects the save location and selects the 'Save' button.

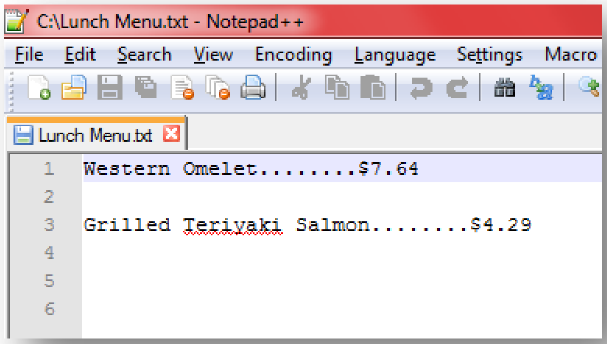


Name	Date modified	Type	Size
_working	9/24/2014 8:53 AM	File folder	
0ed94309d30753d5089ace71f1b286	9/27/2012 3:22 PM	File folder	
apps	9/27/2012 3:53 PM	File folder	
Dell	9/27/2012 12:11 PM	File folder	
git	6/29/2014 11:32 PM	File folder	
MSOCache	9/28/2012 1:43 PM	File folder	
OneDriveTemp	9/29/2014 10:27 PM	File folder	
PerfLogs	7/13/2009 10:20 PM	File folder	
Program Files	5/30/2014 1:03 AM	File folder	
Program Files (x86)	11/13/2014 1:25 PM	File folder	
ProgramData	5/30/2014 12:57 AM	File folder	
temp	6/30/2013 7:46 PM	File folder	
Users	10/1/2012 1:30 PM	File folder	
Windows	10/26/2014 11:38 ...	File folder	
test.txt	9/24/2014 8:48 AM	Notepad++ Docu...	1 KB

File name: Lunch Menu.txt
Save as type: Menus (.txt) (*.txt)

Figure 6-4:Standard 'Save As…' Dialog provided by the application

<u>Step 5:</u> File is saved as specified location.


C:\Lunch Menu.txt - Notepad++
File Edit Search View Encoding Language Settings Macro
Lunch Menu.txt

```
Western Omelet........$7.64

Grilled Teriyaki Salmon........$4.29



```

Figure 6-5:Screen capture of saved menu file

Appendix A – *'Recipe Helper' Source Code*

This section lists out the source code that was written for the development iterations of the 'Recipe Helper' application.

A.1 RecipeHelper.GUI

The code in this section was written for development of the GUI of the application. These objects compose the view in the MVVM design pattern. The only dependencies within the scope of the application are on the view models and any public interfaces, but there are no dependencies on concrete domain objects thereby satisfying the **Single Responsibility Principle**.

MainWindow (xaml)

```
<Window x:Class="RecipeHelper.GUI.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:RecipeHelper.GUI"
        xmlns:VMs="clr-namespace:RecipeHelper.ViewModel;assembly=RecipeHelper.ViewModel"
        xmlns:sdk="clr-namespace:RecipeHelper.SDK;assembly=RecipeHelper.SDK"
        Title="Reipe Helper" Height="414.667" Width="763.333"
        Background="{DynamicResource {x:Static SystemColors.ControlBrushKey}}">
    <Window.DataContext>
        <VMs:RecipeCatalogViewModel/>
    </Window.DataContext>
    <Window.Resources>
        <DataTemplate x:Key="RecipeItemTemplate" DataType="sdk:IRecipe">
            <TextBlock Text="{Binding Path=Name}"
                       VerticalAlignment="Bottom" HorizontalAlignment="Center"/>
        </DataTemplate>
    </Window.Resources>
    <Grid>
        <Button Name="btnAddRecipe" Content="Add Recipe..." Margin="10,353,0,0" Width="75"
                HorizontalAlignment="Left" VerticalAlignment="Top"
                Click="ButtonAddRecipe_OnClick"/>

        <Button Content="Close" Margin="670,353,0,0" Width="75"
                HorizontalAlignment="Left" VerticalAlignment="Top" />

        <Label Content="Recipes" FontSize="18" FontWeight="Bold" Margin="340,10,341,0"
               HorizontalAlignment="Center" VerticalAlignment="Top" />

        <ListBox x:Name="listBoxRecipes" SelectionMode="Extended"
                 Height="59" Width="724" Margin="21,44,0,0"
                 HorizontalAlignment="Left" VerticalAlignment="Top"
                 SelectionChanged="ListBoxRecipes_OnSelectionChanged"
                 ItemsSource="{Binding Model.Recipes}"
                 ItemTemplate="{StaticResource RecipeItemTemplate}"/>

        <Label Content="Cooking Steps:" Margin="21,138,645,0"
               HorizontalAlignment="Center" VerticalAlignment="Top" />

        <ListBox x:Name="textBlockCookingSteps"
                   Margin="21,169,453,0" Height="133" Width="281"
                   HorizontalAlignment="Center" VerticalAlignment="Top"
                   ItemsSource="{Binding ElementName=listBoxRecipes,
                                         Path=SelectedItem.CookingSteps}"/>


        <Label Content="Ingredients:" Margin="483,138,0,0"
               HorizontalAlignment="Left" VerticalAlignment="Top"/>
        <ListBox x:Name="textBlockIngredients" Height="133" Width="262"
                   Margin="483,169,0,0"
                   HorizontalAlignment="Left" VerticalAlignment="Top"
                   ItemsSource="{Binding ElementName=listBoxRecipes,
                                      Path=SelectedItem.Ingredients}"/>

        <Label Content="Prep Time:" Margin="331,178,0,0"  Width="69"
               HorizontalAlignment="Left" VerticalAlignment="Top"/>
        <TextBlock Margin="405,183,0,0" TextWrapping="Wrap" Width="48"
                   HorizontalAlignment="Left" VerticalAlignment="Top"
                   Background="{DynamicResource {x:Static SystemColors.WindowBrushKey}}"
                   Text="{Binding SelectedItem.PrepTime,
                                  ElementName=listBoxRecipes,
                                  Converter={local:IntToTimeConverter}}"/>

        <Label Content="Cook Time:" Margin="331,220,0,0" Width="69"
               HorizontalAlignment="Left" VerticalAlignment="Top"/>
        <TextBlock Margin="405,225,0,0" TextWrapping="Wrap" Width="48"
                   HorizontalAlignment="Left" VerticalAlignment="Top"
                   Background="{DynamicResource {x:Static SystemColors.WindowBrushKey}}"
                   Text="{Binding ElementName=listBoxRecipes,Path=SelectedItem.CookTime,
                                  Converter={local:IntToTimeConverter}}"/>

        <Label Content="Price:" Margin="355,264,0,0" Width="38"
           HorizontalAlignment="Left" VerticalAlignment="Top"/>
        <TextBlock Margin="405,269,0,0" TextWrapping="Wrap" Width="48"
           HorizontalAlignment="Left" VerticalAlignment="Top"
           Background="{DynamicResource {x:Static SystemColors.WindowBrushKey}}"
           Text="{Binding SelectedItem.Price, ElementName=listBoxRecipes,
                          StringFormat={}{0:C2}}"/>

        <Button Name="btnCreateMenu" IsEnabled="False" Content="Create Menu..."
                HorizontalAlignment="Left" Margin="340,353,0,0" VerticalAlignment="Top"
                Width="89" Click="ButtonCreateMenu_Click"/>
    </Grid>
</Window>
```

<u>MainWindow (code-behind)</u>

```
public partial class MainWindow
{
    public MainWindow()
    {
        InitializeComponent();
    }

    private void ButtonAddRecipe_OnClick(object sender, RoutedEventArgs e)
    {
        var editRecipeDlg = new EditRecipeDialog
        {
            Owner = this,
        };
        editRecipeDlg.ShowDialog();
    }

    private void ButtonCreateMenu_Click(object sender, RoutedEventArgs e)
    {
        var menuPreviewDlg = new MenuPreviewDialog()
        {
            Owner = this,
            DataContext = new MenuPreviewViewModel(listBoxRecipes.SelectedItems),
        };
        menuPreviewDlg.ShowDialog();
    }

    private void ListBoxRecipes_OnSelectionChanged(object sender,
        SelectionChangedEventArgs e)
    {
        btnCreateMenu.IsEnabled = (listBoxRecipes.SelectedItems.Count > 0);
    }
}
```

MenuPreviewDialog (xaml)

```
<Window x:Class="RecipeHelper.GUI.MenuPreviewDialog"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:sdk="clr-namespace:RecipeHelper.SDK;assembly=RecipeHelper.SDK"
        xmlns:local="clr-namespace:RecipeHelper.GUI"
        Title="Menu Preview" Height="586" Width="542"
        ResizeMode="NoResize" ShowInTaskbar="False" WindowStartupLocation="CenterOwner">
    <Window.Resources>
        <DataTemplate x:Key="RecipeItemTemplate" DataType="sdk:IRecipe">
            <TextBlock Text="{Binding Path=Name}"/>
        </DataTemplate>
    </Window.Resources>
    <Grid>
        <Grid Background="LemonChiffon" HorizontalAlignment="Left" Height="557"
              VerticalAlignment="Top" Width="536">

            <RichTextBox Name="richTextBoxMenu" HorizontalAlignment="Left" Height="199"
                         Margin="10,44,0,0" VerticalAlignment="Top" Width="516">
                <FlowDocument>
                    <Paragraph>
                        <Run Text="{Binding Path=MenuRecipes,
                                        Converter={local:RecipeListToStringConverter}}"/>
                    </Paragraph>
                </FlowDocument>
            </RichTextBox>

            <Button Name="btnPrintGroceryList" Content="Print Grocery List..."
                    IsEnabled="{Binding ElementName=groceryListRun,Path=Text,
                                    Converter={local:GroceryListToBoolConverter}}"
                    HorizontalAlignment="Left" Margin="216,526,0,0"
                    VerticalAlignment="Top" Width="110"
                    Click="ButtonPrintGroceryList_Click"/>

            <GroupBox Header="Grocery List" HorizontalAlignment="Left"
                      Margin="10,258,0,0" VerticalAlignment="Top"
                      Height="250" Width="516">
                <Grid HorizontalAlignment="Left" Height="229" VerticalAlignment="Top"
                      Width="506" Margin="0,0,-2,-1">

                    <ListBox Name="listBoxRecipes"
                             SelectedItem="{Binding SelectedRecipe}"
                             ItemsSource="{Binding MenuRecipes}"
                             ItemTemplate="{StaticResource RecipeItemTemplate}"
                             HorizontalAlignment="Left" Height="167" Margin="10,52,0,0"
                             VerticalAlignment="Top" Width="154"/>

                    <Label Content="Frequency Factor" HorizontalAlignment="Left"
                           Margin="186,87,0,0" VerticalAlignment="Top"/>

                    <TextBox Text="{Binding FrequencyFactor}" HorizontalAlignment="Left"
                             Height="23" Margin="208,119,0,0" TextWrapping="Wrap"
                             VerticalAlignment="Top" Width="52"/>
                    <Button Name="btnRtArrow" Command="{Binding AddIngredientsCommand}"
                            Content="=&gt;" HorizontalAlignment="Left"
                            Margin="300,120,0,0" VerticalAlignment="Top" Width="28"/>

                    <RichTextBox Name="richTxtBoxGroceryList" HorizontalAlignment="Left"
                                 Height="167" Margin="342,52,0,0" VerticalAlignment="Top"
                                 Width="154">
                        <FlowDocument>
                            <Paragraph>
                                <Run Name="groceryListRun"
                                     Text="{Binding Path=GroceryList,
                                       Converter={local:GroceryListToStringConverter}}"/>
                            </Paragraph>
                        </FlowDocument>
                    </RichTextBox>

                    <Label Content="Recipes" HorizontalAlignment="Left"
                           Margin="60,21,0,0" VerticalAlignment="Top"/>

                    <Label Content="Grocery List" HorizontalAlignment="Left"
                           Margin="383,21,0,0" VerticalAlignment="Top"/>

                    <Label Content="X" HorizontalAlignment="Left" Margin="186,118,0,0"
                           VerticalAlignment="Top"/>
                </Grid>
            </GroupBox>

            <Button Content="Save Menu..." Click="ButtonSaveMenu_OnClick"
                    HorizontalAlignment="Left" Margin="10,526,0,0"
                    VerticalAlignment="Top" Width="80"/>

            <Button Content="Close" HorizontalAlignment="Left" Margin="451,526,0,0"
                    VerticalAlignment="Top" Width="75"/>

            <Label Content="Menu:" HorizontalAlignment="Left" Margin="10,13,0,0"
                   VerticalAlignment="Top" Width="42"/>

            <TextBox Name="textBoxMenuName" HorizontalAlignment="Left" Height="23"
                     Margin="57,15,0,0" TextWrapping="Wrap" VerticalAlignment="Top"
                     Width="280"/>
        </Grid>
    </Grid>
</Window>
```

<u>MenuPreviewDialog (code-behind)</u>

```
public partial class MenuPreviewDialog
{
    public MenuPreviewDialog()
    {
        InitializeComponent();
    }

    private void ButtonPrintGroceryList_Click(object sender, RoutedEventArgs e)
    {
        var printDlg = new PrintDialog();

        var userCancelled = !printDlg.ShowDialog().Value;
        if (userCancelled) return;

        IDocumentPaginatorSource doc = richTxtBoxGroceryList.Document;
        printDlg.PrintDocument(doc.DocumentPaginator,
            "Recipe Helper - Print Grocery List");
    }

    private void ButtonSaveMenu_OnClick(object sender, RoutedEventArgs e)
    {
        var saveDlg = new SaveFileDialog()
        {
            FileName =
                (!string.IsNullOrWhiteSpace(textBoxMenuName.Text)
                    ? textBoxMenuName.Text
                    : "My Menu"),
            DefaultExt = ".txt",
            Filter = "Menus (.txt)|*.txt",
        };

        var saveDialogShown = saveDlg.ShowDialog();

        var userCancelled = (saveDialogShown.HasValue) && (!saveDialogShown.Value);
        if (userCancelled) return;

        Save(saveDlg.FileName);
    }

    private void Save(string fileName)
    {
        var range = new TextRange(richTextBoxMenu.Document.ContentStart,
            richTextBoxMenu.Document.ContentEnd);
        var fStream = new FileStream(fileName, FileMode.Create);
        range.Save(fStream, DataFormats.Text);
        fStream.Close();
    }
}
```

<u>EditRecipeDialog (xaml)</u>

```
<Window x:Class="RecipeHelper.GUI.EditRecipeDialog"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Edit Recipe" Height="501" Width="605" Background="LemonChiffon"
        ResizeMode="NoResize" ShowInTaskbar="False" WindowStartupLocation="CenterOwner">
    <Grid>
        <Button Content="Save" HorizontalAlignment="Left" Margin="10,439,0,0"
                VerticalAlignment="Top" Width="75"/>
        <Button Content="Cancel" HorizontalAlignment="Left" Margin="512,439,0,0"
                VerticalAlignment="Top" Width="75"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="58,33,0,0"
                 TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top" Width="217"/>
        <Label Content="Name" HorizontalAlignment="Left" Margin="10,33,0,0"
               VerticalAlignment="Top" Width="43"/>
        <GroupBox Header="Add Ingredients" HorizontalAlignment="Left" Margin="10,64,0,0"
                  VerticalAlignment="Top" Height="190" Width="577">
            <Grid HorizontalAlignment="Left" Height="157" Margin="10,10,-2,0"
                  VerticalAlignment="Top" Width="557">
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="238*"/>
                    <ColumnDefinition Width="57*"/>
                    <ColumnDefinition Width="262*"/>
                </Grid.ColumnDefinitions>
                <Button Content="=&gt;" Grid.Column="1" HorizontalAlignment="Left"
                        Margin="10,61,0,0" VerticalAlignment="Top" Width="38" Height="22"/>
                <ListBox Grid.Column="2" HorizontalAlignment="Left" Height="137"
                         Margin="10,10,0,0" VerticalAlignment="Top" Width="242"/>
                <Label Content="Quantity:" HorizontalAlignment="Left"
```

```
                       VerticalAlignment="Top" Width="60" Height="26"/>
                <TextBox HorizontalAlignment="Left" Height="23" Margin="65,2,0,0"
                         TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top"
                         Width="71"/>
                <Label Content="Units:" HorizontalAlignment="Left" Margin="0,61,0,0"
                       VerticalAlignment="Top" Width="60" Height="26"/>
                <ComboBox HorizontalAlignment="Left" Margin="65,61,0,0"
                          VerticalAlignment="Top" Width="126" Height="22"/>
                <Label Content="Name:" HorizontalAlignment="Left" Margin="0,121,0,0"
                       VerticalAlignment="Top" Width="60" Height="26"/>
                <TextBox HorizontalAlignment="Left" Height="23" Margin="65,123,0,0"
                         TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top"
                         Width="140"/>
            </Grid>
        </GroupBox>
        <GroupBox Header="Cooking Steps" HorizontalAlignment="Left" Margin="10,259,0,0"
                  VerticalAlignment="Top" Height="175" Width="577">
            <Grid HorizontalAlignment="Left" Height="140" Margin="10,10,-2,0"
                  VerticalAlignment="Top" Width="557">
                <Label Content="Prep Time (minutes)" HorizontalAlignment="Left"
                       Margin="410,10,0,0" VerticalAlignment="Top" Width="119"/>
                <TextBox HorizontalAlignment="Left" Height="23" Margin="446,41,0,0"
                         TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top"
                         Width="55"/>
                <Label Content="Cook Time (minutes)" HorizontalAlignment="Left"
                       Margin="410,76,0,0" VerticalAlignment="Top" Width="119"/>
                <TextBox HorizontalAlignment="Left" Height="23" Margin="446,107,0,0"
                         TextWrapping="Wrap" Text="TextBox" VerticalAlignment="Top"
                         Width="55"/>
                <RichTextBox HorizontalAlignment="Left" Height="120" Margin="10,10,0,0"
                             VerticalAlignment="Top" Width="323">
                    <FlowDocument>
                        <Paragraph>
                            <Run Text="RichTextBox"/>
                        </Paragraph>
                    </FlowDocument>
                </RichTextBox>
            </Grid>
        </GroupBox>
    </Grid>
</Window>
```

EditRecipeDialog (code-behind)

```
public partial class EditRecipeDialog
{
    public EditRecipeDialog()
    {
        InitializeComponent();
    }
}
```

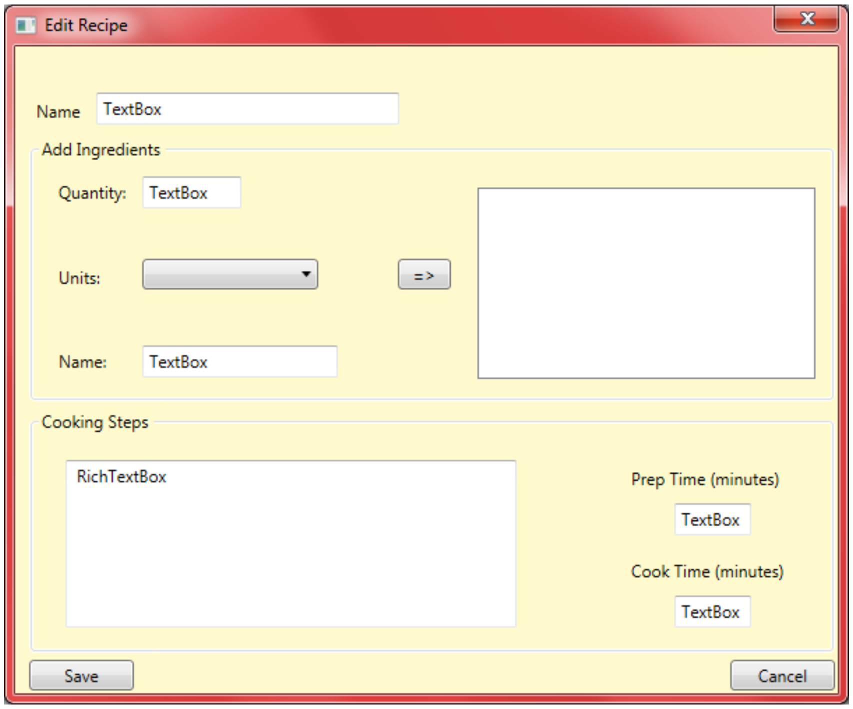


Figure A-1: Edit Recipe Dialog

A.2 RecipeHelper.Library

The code in this section represents the primary domain objects required for the iterations of development presented in this project. More development iterations would have inevitably led to the need for more objects in order to define concrete realizations of the interfaces in the RecipeHelper.SDK library.

RecipeCatalog

```
public class RecipeCatalog
{
    public ObservableCollection<IRecipe> Recipes { get; set; }

    public RecipeCatalog()
    {
        Recipes = Utilities.LoadRecipes();
    }
}
```

A.3 RecipeHelper.SDK

The code in this section represents the objects depended on by the other libraries. All public interfaces are defined in this library and helps encourage compliance with the **Liskov Substitution Principle**.

IRecipe

```
public interface IRecipe
{
    string Name { get; set; }
    List<IIngredient> Ingredients { get; set; }
    List<string> CookingSteps { get; set; }
    int PrepTime { get; set; }
    int CookTime { get; set; }
    double Price { get; set; }
}
```

Ingredient

```
public interface IIngredient
{
    int Quantity { get; set; }
    Unit Unit { get; set; }
    string Name { get; set; }
}
```

Unit

```
public enum Unit
{
    Gram,
    Ounce,
    Pound,
}
```

A.4 RecipeHelper.Utils

This library is intended to be utilized by all other libraries requiring common utility methods such as external database and framework interactions. This library helps with comply with the **Don't Repeat Yourself Principle**. This library was also used to encompass the mock objects required to complete the development iterations. These mock objects served as stand-ins for what would be objects pulled in from a database.

Utilities

```
public class Utilities
{
    public static ObservableCollection<IRecipe> LoadRecipes()
    {
        return new ObservableCollection<IRecipe>
                {
                    new MockRecipe("Western Omelet", 60),
                    new MockRecipe("Mashed Potatoes", 90),
                    new MockRecipe("Grilled Teriyaki Salmon", 90),
                };
    }
}
```

MockRecipe

```
internal class MockRecipe : IRecipe
{
    private static readonly Random Random = new Random();

    public MockRecipe(string name, int seed = 0)
    {
        Name = name;
        PrepTime = seed;
        CookTime = seed;
        Ingredients = new List<IIngredient>
                    {
                        new MockIngredient(string.Format("{0}-Ingredient {1}", Name, 1)),
                        new MockIngredient(string.Format("{0}-Ingredient {1}", Name, 2)),
                    };
        CookingSteps = new List<string>
                    {
                        string.Format("{0}-Step {1}:", Name, 1),
                            string.Format("{0}-Step {1}:", Name, 2)
                        };
        Price = Random.Next(1, 10) + Random.NextDouble();
    }

    public string Name { get; set; }
    public List<IIngredient> Ingredients { get; set; }
    public List<string> CookingSteps { get; set; }
    public int PrepTime { get; set; }
    public int CookTime { get; set; }
    public double Price { get; set; }
}
```

MockIngredient

```
internal class MockIngredient : IIngredient
{
    public MockIngredient(string name)
    {
        Name = name;
        Quantity = 1;
        Unit = Unit.Gram;
    }

    public int Quantity { get; set; }
    public Unit Unit { get; set; }
    public string Name { get; set; }

    public override string ToString()
    {
            return string.Format("{0} {1} {2}", Quantity,
                Enum.GetName(typeof (Unit), Unit), Name);
    }
}
```

A.5 RecipeHelper.ViewModel

The code in this section represents the view model objects of the MVVM design pattern. They help reduce coupling and dependencies between the GUI and domain objects.

ViewModelBase

```
public abstract class ViewModelBase : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    [NotifyPropertyChangedInvocator]
    protected virtual void OnPropertyChanged(
        [CallerMemberName] string propertyName = null)
    {
        var handler = PropertyChanged;
        if (handler != null)
            handler(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

RecipeCatalogViewModel

```
public class RecipeCatalogViewModel : ViewModelBase
{
    public RecipeCatalog Model { get; private set; }

    public RecipeCatalogViewModel()
    {
        Model = new RecipeCatalog();
    }
}
```

MenuPreviewViewModel

```
public class MenuPreviewViewModel : ViewModelBase
{
    private Dictionary<IRecipe, int> FrequencyFactors { get; set; }
    public IList<IRecipe> MenuRecipes { get; set; }
    public DelegateCommand AddIngredientsCommand { get; private set; }

    private IRecipe _selectedRecipe;
    public IRecipe SelectedRecipe
    {
        get { return _selectedRecipe; }
        set
        {
            _selectedRecipe = value;
            OnPropertyChanged();
            OnPropertyChanged("FrequencyFactor");
        }
    }

    public int FrequencyFactor
    {
        get
        {
            return (SelectedRecipe != null) ? FrequencyFactors[SelectedRecipe] : 1;
        }
        set
        {
            FrequencyFactors[SelectedRecipe] = value;
            OnPropertyChanged();
        }
    }

    public ObservableCollection<IIngredient> GroceryList { get; set; }

    private void UpdateGroceryList()
    {
        for (int n = 0; n < FrequencyFactors[SelectedRecipe]; n++)
        {
            SelectedRecipe.Ingredients.ForEach(i => GroceryList.Add(i));
        }
        OnPropertyChanged("GroceryList");
    }

    public MenuPreviewViewModel(IEnumerable menuRecipes)
    {
        MenuRecipes = menuRecipes.Cast<IRecipe>().ToList();
        SelectedRecipe = MenuRecipes.First();
        FrequencyFactors = new Dictionary<IRecipe, int>(MenuRecipes.Count);
        foreach (var recipe in MenuRecipes)
        {
            FrequencyFactors.Add(recipe, 1);
        }
        AddIngredientsCommand = new DelegateCommand(UpdateGroceryList);
        GroceryList = new ObservableCollection<IIngredient>();
    }
}
```

Non-Direct Activity Report

Date	Duration (minutes)	Specific Task / Activity
27-Oct-2014	45	Project submittal/cleanup/finalization
2-Nov-2014	50	Project #3 setup & initialization
4-Nov-2014	60	Work on Project #3
9-Nov-2014	122	Work on Project #3
11-Nov-2014	114	Work on Project #3
13-Nov-2014	60	Work on Project #3
16-Nov-2014	205	Work on Project #3
18-Nov-2014	19	Work on Project #3
23-Nov-2014	467	Work on Project #3
24-Nov-2014	55	Work on Project #3
29-Nov-2014	210	Work on Project #3
30-Nov-2014	480	Work on Project #3
1-Dec-2014	79	Work on Project #3
2-Dec-2014	68	Work on Project #3
4-Dec-2014	360	Work on Project #3
5-Dec-2014	600	Work on Project #3
Sum for Report #1	**2513**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Report #2	**1938**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Report #3	**2994**	**/ 1500 (5 weeks @ 300/wk)**
Sum for Class	**7445**	**/ 4500 (15 weeks @ 300/wk)**