*Due:* Monday, May 1st *Worth:* 20% of course grade

Contents

[Analysis Info 2](#_Toc478557245)

[Scenario 2](#_Toc478557246)

[Business Type 2](#_Toc478557247)

[Pair Programming 2](#_Toc478557248)

[Use Case Diagram 3](#_Toc478557249)

[Domain Class Diagram 4](#_Toc478557250)

[Sample Values for Types 4](#_Toc478557251)

[Use Case Descriptions 5](#_Toc478557252)

[Viewing Lists 5](#_Toc478557253)

[Maintaining Records 5](#_Toc478557254)

[Specialized Features 6](#_Toc478557255)

[Design & Programming Requirements 7](#_Toc478557256)

[Getting Started 7](#_Toc478557257)

[Start Coding 8](#_Toc478557258)

[Domain Classes 8](#_Toc478557259)

[File Load 8](#_Toc478557260)

[Code the Use Cases 9](#_Toc478557261)

[File Save 9](#_Toc478557262)

[Do’s & Don’t 9](#_Toc478557263)

[Forms 9](#_Toc478557264)

[Files 9](#_Toc478557265)

[DataGridView 9](#_Toc478557266)

[Data Requirements 10](#_Toc478557267)

[Auto Load Data 10](#_Toc478557268)

[GUI Requirements 11](#_Toc478557269)

[Other Requirements 12](#_Toc478557270)

[Submitting the Assignment 13](#_Toc478557271)

[Presentation 14](#_Toc478557272)

[Requirements 14](#_Toc478557273)

[Documents 14](#_Toc478557274)

[Presentation Attendance 14](#_Toc478557275)

[Peer Evaluations 14](#_Toc478557276)

# Analysis Info

## Scenario

You are to design a user-friendly customer complaint system that allows users (employees) to enter and track customer complaints. The application is a standalone application and thus must capture/update the customer’s personal profile. Next, the user will enter the complaint details. The system should then provide the ability to view statistics related to the complaints in a form that would be useful to management.

## Business Type

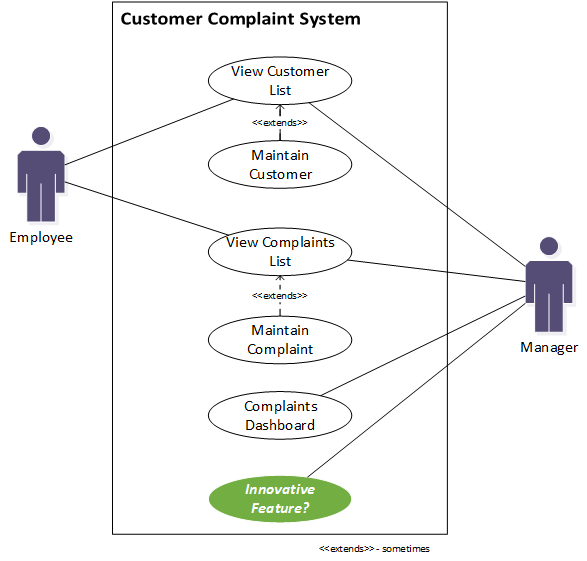
Think of a business that has customers that may issue complaints. Design your entire application with that business in mind. Each team should seek to do a different business.

## Pair Programming

I recommend that you employ *Pair Programming* in this project. Each pair will develop code together at one computer. There will be one *driver* (typing the code) and the other is the *navigator* (pointing out issues and making suggestions). The driver is expected to “program out loud” – i.e. talk about what he/she is doing while coding. The navigator should be actively engaged as well. The goal is to take turns being driver and navigator. The benefit of this approach is that, when done correctly, it should increase code quality and knowledge transfer (both in the problem domain as well as in programming techniques). Pair Programming is a feature of XP development and may also be used in other agile approaches.

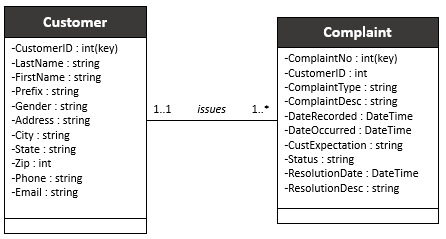
## Use Case Diagram

Below is a UML Use Case Diagram with all use cases that your client is requesting. While actors have been denoted, do not worry about designing a login nor limiting features. Develop this such that the manager has access to all use cases.



## Domain Class Diagram

Below is a UML Domain Class Diagram with the classes involved in this project. You may add to this diagram by adding additional attributes and classes, where appropriate. Do not override any of the existing fields nor their defined data types.



### Sample Values for Types

This might be a set of values for a Dry-Cleaning business.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Prefix** |  | **Complaint Type** |  | **Customer Expectation** |  |
| Mr |  | Stain |  | Service Refund |  |
| Mrs |  | Tear |  | Item Refund |  |
| Dr |  | Button Missing |  | Repair |  |
| … |  | … |  | … |  |

## Use Case Descriptions

### Viewing Lists

Assign a pair to work on the following 2 use cases and employ pair programming. Complete the 1st use case in its entirety before going to the second one as they are almost identical. When completed both use cases should have the same interface design.

#### View Customers List

* Allow the user to view a *list of all* customers. You must use a dataGridView to display the customers and set its original data source as the customers list. Thus, all fields will be shown. Do not allow any data updates in the data grid. Add capability to do a single-field *sort* utilizing the header field names in the data grid. Using a different GUI control, add capability to do a multi-field sort. Add capability to do *record* *navigation* (first, previous, next, last) in the data grid. Add capability to *filter* this data (i.e. view a shorter list) on a minimum of 4 fields. This should allow both single-field or multi-field filtering. Add a clear filter button to reset and display all data.

#### View Complaints List

* Allow the user to view a *list of all* complaints. The rest of the capabilities are the same as documented in the *View Customer List* use case. An additional requirement is that one of the filter options must be a date field.

### Maintaining Records

Assign a pair to work on the following 2 use cases and employ pair programming. Complete the 1st use case in its entirety before going to the second one as they are almost identical. When completed both use cases should have the same interface design.

#### Maintain Customer

* Allow the user to *view* an individual customer’s profile on one form. The form should be designed like a traditional data entry form (with fields listed vertically). You must use appropriate GUI controls for each field – i.e. a Monthly Calendar or DateTime Picker for dates, etc. Since there are many customer fields, ensure that you group related fields. In addition, the user should be able to *update* and *delete* an existing customer or *add* a new customer on this form. All data changes must be reflected in the customer list. Include navigation (first, previous, next, last). Add a clear button to clear data from any input controls (e.g. textboxes, etc.). You may not remove a field but you may need to add fields to the Customer class to support other features.

#### Maintain Complaint

* Allow the user to *view* an individual complaint profile on one form. The rest of the capabilities are the same as documented in the *Maintain Customer* use case. An additional requirement is to display non-updateable labels with the customer’s ID & name on this form.

### Specialized Features

#### View Complaints Dashboard

* Design a single dashboard that displays a minimum of 8 key performance indicators regarding complaints. Some KPIs may be individual values (e.g. total # of complaints) but you must have a minimum of 3 charts and utilize at least 2 chart styles (pie, column etc.). *All stats* must have a visual design (color, graphs, progress bars, indicators, larger fonts & color, etc.). The dashboard should load/fill automatically when the user opens it and display stats utilizing the most current data. Charts must be generated using the Chart control in the Visual Studio ToolBox – see my special topics links to view the notes on Charts. Add capability to filter on at least one factor; your dashboard should auto-update when a filter is applied. Add capability to hover on at least one statistic and display additional details related to that stat in the form or some type of pop up (but not a new form). Check out idashboards.com for data ideas.

#### Innovative Feature

* Identify one additional (and distinct) feature that sets your product apart from others. This feature should be truly innovative. It must involve you writing significant code, comparable to the other use cases in complexity but different from the other use cases. Do not design more analytics as this is covered in the dashboard. **Your idea must include additional attributes to the 2 classes (or add new a class) to support this feature.** Please note that a feature is more than a single new attribute. The goal is a single excellent idea - not several mediocre ideas. I have a webpage containing a list of special topics (and code) on Canvas that you might useful – if you incorporate something from that list, you must add more functionality than simply copying that code. If you are not sure if your idea is useful and innovative enough, feel free to ask.

# Design & Programming Requirements

## Getting Started

* 1. **Name your Visual Studio project:** *ComplaintSys*. Do not include a space in the project name.
  2. **Rename Form1 to MainForm.** Do not include a space in the form name.
  3. **Interface Design** – sketch a wireframe showing the layout and style of your application. Look over the idea & get agreement as a team before going forward. Decide the following:
     1. Navigation approach – decide how the user will navigate: tabControl, menuStrip, icons, pictureBoxes, toolStrip, etc.
     2. Form size – pick a standard size for your entire application. You might make it large enough so it can accommodate your Dashboard.
     3. Layout, Icons/Buttons – design a consistent layout, icons & buttons, etc.
     4. Color Scheme – every form must look modern & professional, not just your MainForm, Dashboard, and AboutForm.
     5. Design the interface(s) for each use case.
  4. **Use the 2 csv files.** I provided these starter files plus a generator in case you need more records. The CSV files should not include a 1st row with field names or your program will crash. The fields must always be in the order that they appear in the Domain Class code.

## Start Coding

### Domain Classes

* Write the 2 Domain Classes. Make the class *public* so it’s accessible in other classes. Code all fields as *auto-implemented properties* unless you have additional code to add to the properties. Code the properties in the order that they appear in the Domain Class Diagram above for consistency. Make sure your domain classes are *public*. Only one version of the customer class & complaint class is allowed throughout the project.

### File Load

Goal: Pre*-*load the customer file into the Customer List, and load the complaints file into the Complaints list.

* *Declare your lists* **-** declare and instantiate yourcustomers & complaints lists as fields in your MainForm class (and make them public) so they are accessible by all methods and in all forms. Additional forms can access these: *MainForm.customers & MainForm.complaints.*
* *Load the CSV files into a List<T>* - write a method called **LoadCustomersFromFile**() to load your customers list from the CSV file. This method does not display data in a grid as that happens elsewhere. This is only loading the data into a List<Customer> in memory – you have to instantiate the customer objects and add them to the list. You can write something temporarily to a label just to test that it is working. Common issues are not handling empty values or having data that doesn’t match the data types in the domain class. Make sure this method is called when the MainForm is Loaded. Once this is working perfectly, repeat the same logic and write a method called **LoadComplaintsFromFile**(). The complaints method will have an added task of dealing with date fields.
* I provided 2 starting files plus a data generation tool for the complaints file. You may use that generator if you need more complaint records. Place the 2 CVS files in your project’s bin\debug folder. Also, maintain a backup copy of those files in there as well: BKUP-Customers.csv & BKUP-Complaints.csv

### Code the Use Cases

Goal: Each use case will use the lists (*MainForm.customers & MainForm.complaints)* that were loaded as soon as the application runs. That is, you will not be accessing the CSV files. As such, when an update occurs in one use case, that change is reflected in another one since all use cases are accessing the same lists.

* Viewing Lists – 2 use cases that are similar
* Maintaining Records – 2 use cases that are similar
* View Complaints Dashboard
* Innovative Features

### File Save

Goal: Provide the capability to save the lists back to the 2 CSV files: customer & complaints.

## Do’s & Don’t

### Forms

* Name the first form: *MainForm*
* Every form name must be descriptive and have the word *Form* as the last word: About*Form*
* Please do not design a Login Form!
* No spaces in Form names, file names, or project names
* Center the form on the screen.

### Files

* Include 2 backup files w/full test data: BKUP-Customers.csv & BKUP-Complaints.csv
* Must be able to handle missing dates or numbers when saving or opening a file.

### DataGridView

* The user should not be able to update values in a data grid.
* All data grid values should be properly aligned: text-left, numbers-right, dates-right, and codes or any values that are a same fixed length should be centered.
* The main columns on the left should be frozen as the user scrolls to the right and it the frozen columns should have a distinct but subtle color.
* Data should be formatted appropriately: dollar figures ($200.00), dates (01/31/2020),

## Data Requirements

* *Data Validation* – data must be validated on input
* *Persistent data* – all data must be loaded from & saved to csv file(s) and available on the next execution.
* *Dates* – all dates must be defined as DateTime objects and not as strings. See my Special Topic on DateTime. Some dates will be empty - if a date is empty on the CSV file, do not load it or it will crash. All dates must be stores as a DateTime object and must be displayed in mm/dd/yyyy format. Do not show a time with the date.
* *Phone number* *& Check Number* – I chose a string data type for these so it can accommodate a number with leading zeroes - e.g. (010)-555-1111, or check #: 0001. However, I do not want you to store any formatting characters such as parenthesis and hyphens. Make sure these consists only of numbers. You should format it when it is displayed.
* *Totals* – Display the total number of records anywhere where there is a list. If the list is filtered, the total number of records meeting the criteria should be displayed.

## Auto Load Data

* Ensure that your application automatically loads both the customer profiles and complaints for at least 50 customers and 100 complaints. After loading the data, the system should allow the user to add additional customers or complaints. Note: If you have added other data (i.e. additional attributes or classes) in your application, it should load that data as well. The idea is that one can simply open the software and have all of the data necessary to run all features in this application. If any changes are made during the use of the application, those changes should be saved an auto loaded when the application is re-opened.

## GUI Requirements

**Note**: Do not copy any design that we used in class. You must design a modern desktop interface. Many of the interfaces used in class were purely to teach coding and were not intended to be an example of great interface design (e.g. Multiple Sort buttons – Sort LN, Sort PK). You may check Microsoft website as well as other for modem desktop design examples.

* *System Name*
  + identify a unique, marketing name for your application
* *GUI Design*
  + must be innovative, using icons/images for buttons, great color scheme, and not a standard gray application. Look at other applications online for ideas; design something better than competitors.
* *Titles* – all forms must have the System name in the title as well as the *purpose*. For example:  
  XYZ Software - Inventory Analysis
* *Tab Order* - ensure that the tab order is correct on the form controls
* *Data entry form*s - should look like a data entry form. Do not use a data grid for data entry as it is very prone to user errors. That is, design a form where we can only enter one record*. Capability* must include: add, update, delete, view as well as record navigation (first, previous, next, last), clear form (to clear textboxes, etc). You may use a data grid but only for viewing data only.
* *Add an About Form:*
  + Design a *professional looking* About Form & include the following:
    - *Software name*: your software name
    - *Features:* list of the major features (and denote the major innovation)
    - *Developers*: Last name, First name (list all members, denote the Project Leader)
    - *Semester*: Spring 2017
* *Controls -* must include the following:
  + *Panels/Group Boxes* - Use these container controls to group areas on your form for readability
  + *Menu Strip* - minimum: File |Close, Help | About
  + *Tab Control*
  + *PictureBox*
  + *DataGridView*
  + *MonthCalendar or DateTime Picker* – for all dates and/or times
  + *Other controls (where appropriate)* – radio buttons, checkboxes, list boxes, combo boxes, numeric up down, etc.
  + Additional control types are optional

## Other Requirements

* *Show use of the following C# components:*
  + Domain Classes – fields, properties, methods
  + Loops, Decisions, Methods
  + Lists – a list of objects
* *Add Comment Lines* 
  + Add these comments at the top of Form1.cs – or whatever the 1st form that is displayed. (required)
    - *Software name*: your software name
    - *Description*: description/purpose of this software application written in your own words
    - *Semester*: Spring 2017
    - *Class*: MWF ??:??
    - *Team #*: ??
    - *Developers*: Last name, First name (list all customers, denote the Project Leader)
    - ***Additional Functionality***: state what the feature is and where it is located
  + Add XML Comments above each method you created. Include parameter comments.
  + Add additional comments in your code, as necessary
* *Achieve superior Software Quality*
  + **Functionality** – provide all requested functionality & features
  + **Usability** – make your interface intuitive, easy to use, readable, perfect alignment, no typos,…
  + **Reliability** – provide accurate results
  + **Maintainability** – write code that is easy to update: follow standard naming conventions, no repetitive code, no breaking out of programming logic, use methods, add comments, etc.)
  + **Robustness** – write code that doesn't crash!
  + **Performance** – write code that is efficient

# Submitting the Assignment

* *Do the following when you have completed your project:*

1. **This should be submitted by the project leader only.**
2. **Create a backup** of your original project solution folder for your protection.
3. **Zip** (i.e. compress) **your project solution folder**. To zip a folder in Windows, *right click the folder | Send to | Compressed (zipped) folder.* 
   * + - Name your *zip file* as follows: *MW###-T#-Leader.zip* where MW### is either MW125 or MW230 for your class time, T# is your team number, and Leader is the team leader’s last name (e.g. MWF125-T1-Doe)
4. In Canvas, go to *Modules | Homework Assignments | select TeamProj* and **upload the** **zipped file**. *Note:* It will only accept files that end in \*.zip You may upload your project as many times as you want as long it’s before the due date & time. The last file that was uploaded (on time) is the only one that will be graded.
5. **Test that you uploaded your project correctly**.It is your responsibility to upload your project correctly. **A project that cannot be opened will receive a grade of 0** – again, it is the student’s responsibility to make certain that their assignment was uploaded correctly.
   * + - You can test if it was uploaded correctly as follows:
         * Download your uploaded zipped file from Canvas
         * Extract the zipped file:   
            - in Windows, *right click on the zipped file, select “Extract All”.*
         * Open your project from within the unzipped folder, and see if it runs.   
            **- FYI…** You cannot open a zipped file & run your project directly from   
            inside a zipped folder. You must first extract (i.e. unzip) it.

# Presentation

## Requirements

* The presentation consists only of demonstrating your software. There will be no PowerPoint involved. Introduce your team and immediately begin explaining your system. You must pre-load your data.
* Everyone must present (about an equal amount of time)
* Dress: Business Casual

## Documents

* Provide a stapled, black & white printout of your user interfaces with data included. Print one screen per page. Ensure that all text is readable. Include the standard cover page which is on Canvas. Staple these in the order in which you are going to present your application. This document will be used for me to take notes during your presentation and thus must be provided at the beginning of the presentation.

## Presentation Attendance

* Everyone must attend *all* presentations – 2 points will be deducted from the individual's project grade for *each* presentation missed.

# Peer Evaluations

* All students will be evaluated regarding their individual performance. Evaluation Criteria includes: participated in team discussions, met assigned deadlines, produced quality deliverables, and contributed fairly to team effort.
* All students must submit a Peer Evaluation Form by the project submission date/time. Students will not receive a team assignment grade until they have turned in an evaluation form.