iii) System Design Diagrams

a) Use Case Diagram

1) This diagram conveys which parts of the program a user interacts with. The user is able to load a file and view graphs and data, but unable to interact with how the program prepares this information for the user. The parts of the program the user is not able to interact with is represented in a very abstract and high-level format.

2) The rationale for the design of this diagram is based heavily on the requirements for the program and our design and implementation of the program. The customer's requirements list that the user be able to load a selected file into the program, view data (in +/- expandable list), and view graphs. Each of these direct interactions by the user were visually extended with abstract processes of the program that allowed each interaction to happen.

b) Class Diagram

1) This diagram conveys the design of all classes in the program and how these classes are related to one another. From a high level, we see in the diagram that there is a Main class that is used to launch the MainWindow. From the MainWindow, two different actions are taken. The first action is the work done by the program to parse the ".csv" file, create objects from the rows in this file and subsequently add these objects to a map. The second action is the post processing completed on this map to form objects for the +/- expandable list and graphs.

2) The design for this program was based on the Builder Design developed by The Gang of Four. This allowed use to break the tasks of file parsing and user interface into levels. On the file parsing side, the Director class "pulls the strings" of building row objects from a high level. The Director will parse the ".csv" file and pass each parsed row to a RowBuilder class, which is at a level below it. The RowBuilder is responsible for building the row object by calling the AttributeReceiver which performs one level lower operations by retrieving a specific attribute from a given row. For example, the AttributeRetreiver will remove a name, date, string or integer from a ".csv" row and return it to the RowBuilder so that it can be assigned to a row object member variable. From here, the row object is passed back to the Director and inserted into a map.

   The creation of this map allowed for simpler processing down the line to form objects which could be passed to a generic graphing tool, or generic +/- expandable list tool. The benefit of this strategy was regardless of whichever of the 4 file types being used, one GUI class (either for graphs or the +/- expandable lists) could display

all of them without having specific implementation for each file type. At the implementation level, this was done by the ListBuilder which creates a ListClass object from a row object map and the GraphClass which creates a List<List<BarValue > > *. These two objects are then passed to the PlusMinusList and graph dialog classes respectively for display in the MainWindow.

c) Sequence Diagram for a Scenario

1) This diagram conveys the sequence of interactions between classes leading up to the display of a +/- list of Grants.

2) This process has fundamental 3 parts which were separated to allow the team to split up and work on the project: (a) data parsing (b) data consolidation and (c) Graphic display of data.
In (a) the main window class calls upon a director class (while passing in a raw CSV file) to create a map or Grant Row Objects (which hold all of the details for an individual role). This Director calls upon a Row Builder to combine the attributes from the raw CSV text string. Pulling each cell out of the CSV text string is done by the Attribute Retriever Class. This was broken up like such to facilitate debugging and allow the team to split into smaller groups to create/test the classes. To this end, a single team member often became an 'expert' on their class which smoothed out the debugging process significantly.
In (b) the Row Objects (which contain all of the information from each individual STAR entry) are compiled, tallies are created (for total grant funding for an individual - for example). This step is to save real-time processing power in displaying the +/- list in conjunction with meeting the customer requirement of displaying subtotals. This data is saved to a hierarchal list for easier access by the display functions.
In (c) the data is displayed in the +/- list format (as specified by he customer). Since theta was well organized in step (b), the end user will not experience excessive wait times while interacting with the +/- list.

d) Package Diagram

1) The diagram conveys the classes by grouping them based on their function towards the overall program. Every class falls under a grouping (or package), except for the Main and the Director classes, which stand separately. Arrows connect the packages and portray the hierarchy of the implementation.

2) The rational for the design of this diagram was to group the classes as best as possible by their function in relation to the overall program. The Visual package entails all classes that are specifically used for the user interface aspect of the application. The Builder package and Row Object package encompass the four builders and four row objects respectively, with each builder and row object corresponding to one of the four types of files. The Supporting package contains

the classes that are used in association with the other packages, but don't relate to the other classes enough to be put in the same package. The Main and Director classes stand separately as they do not share the same functionalities as the other packages, and are not Supporting classes either.