**CS3307**

**2016 First Semester**

**Peachy Galaxy Phase 2**

**Team Lynx**

**Haitian Yu, Claire Robichon, Yusi Xu, Ryan Vansteenkiste, Aboudi Absi, Matthew Dudycz, Anthony Giugno**

**Professor Nazim Madhavji**

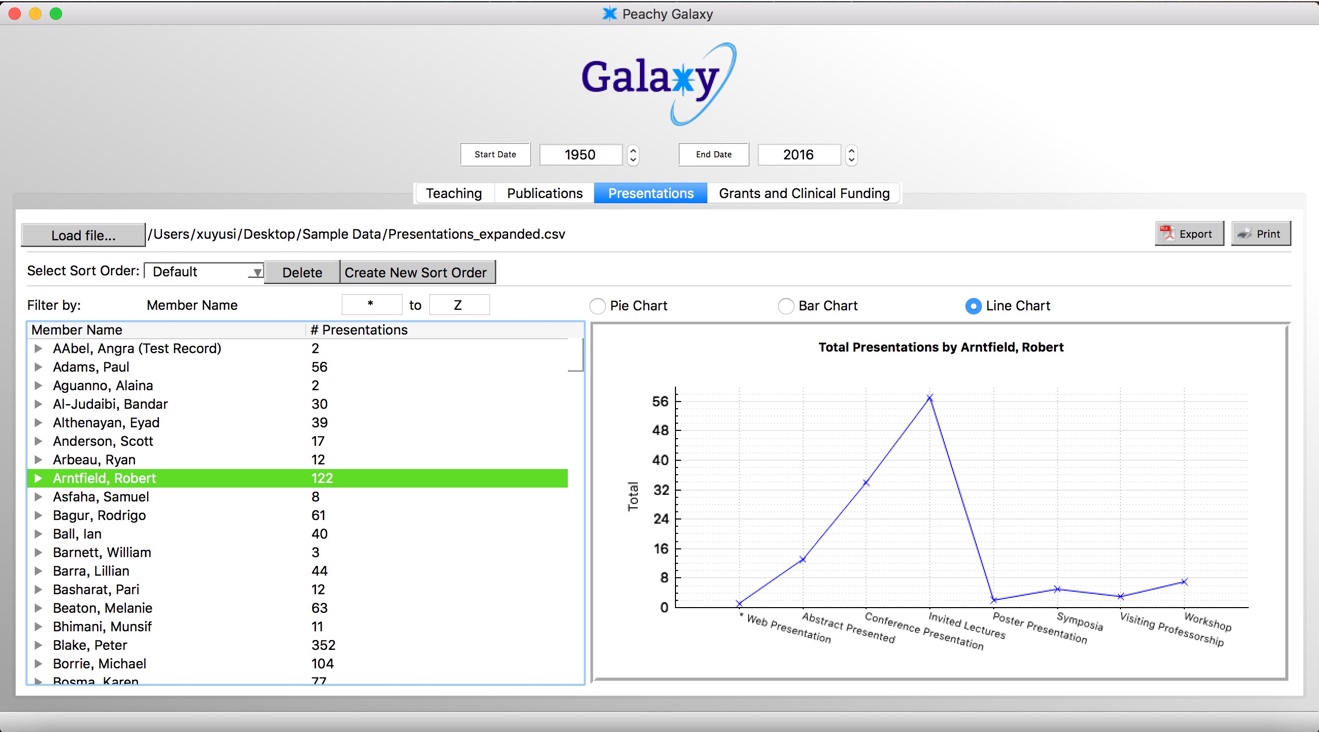
**ID:** MandatoryRequirement2

**Description:** Add a line chart and scatterplot in reporting options

**Origin:** Yusi Xu, Claire Robichon

**Supporting example:**

When we load the file for the Presentations and click on the member name, for example “Arntfield, Robert” here, we choose the graph type “Line Chart” and it will show like following.



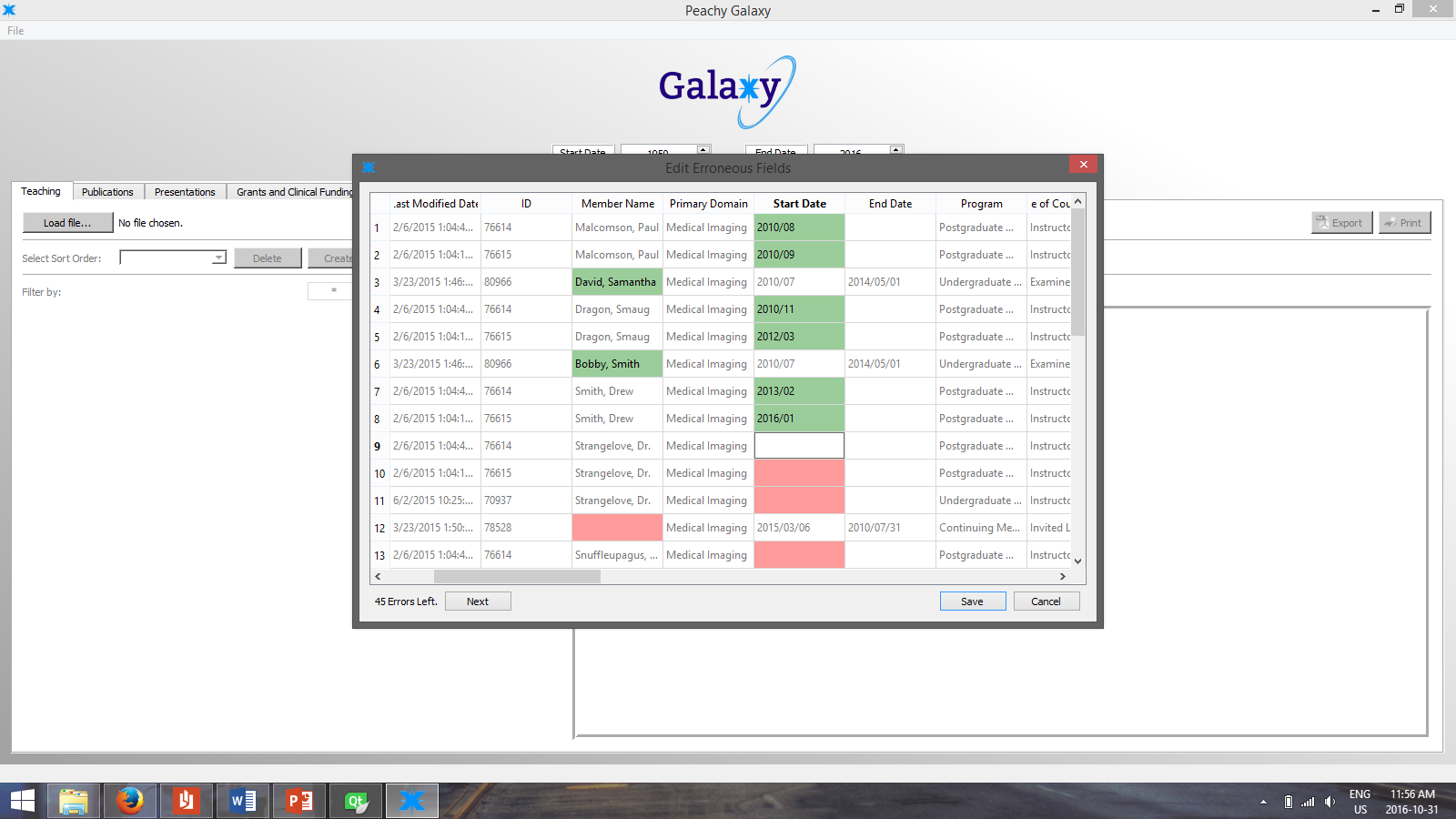
**ID:** MandatoryRequirement9

**Description:** Navigation of Erroneous Entries (Mandatory Requirement)

When the CSV file is loaded for the first time, the error dialog box should report total number of errors and "Find Next/Prev" button which jumps on to the next/prev error. As the errors are fixed, the error count should go down. After all the errors are fixed, "Find Next/Prev" should be disabled.

**Origin:** Code created by Anthony Giugno

**Supporting Example:** An additional feature for user experiences purposes is also shown in the image below, which is the background color for the error cells turning to green when a specific error has been fixed.



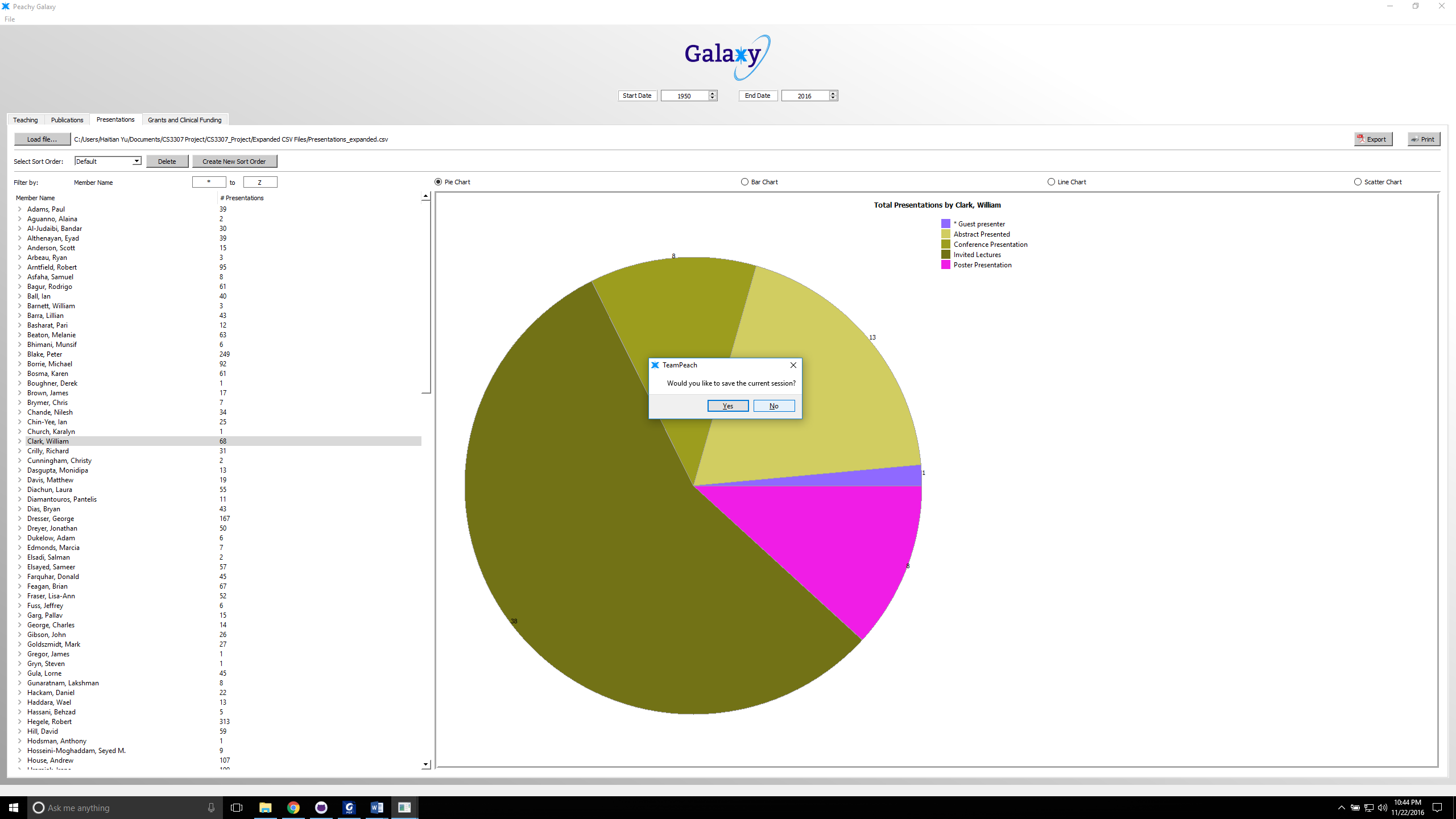
**ID:** MandatoryRequirement3

**Description:** Ability to save session state (Mandatory Requirement)

When the user closes and reopens the application, all the data that was loaded into the program will be saved and re-opened again. The user does not have to re-upload the data.

**Origin:** Code created by Haitian Yu, Anthony Giugno, Claire Robichon

Supporting Example:



**ID:** MandatoryRequirement4

**Description:** Adding the ability to sort by the member’s division

Before, in the drop down menu, there was only the option to select sort order. We have added the option to sort by member’s division.

**Origin:** Code created by Aboudi

**Supporting Example**: **INCOMPLETE**

**ID:** MandatoryRequirement5

**Description:** Adding the ability to sort by a user selected list

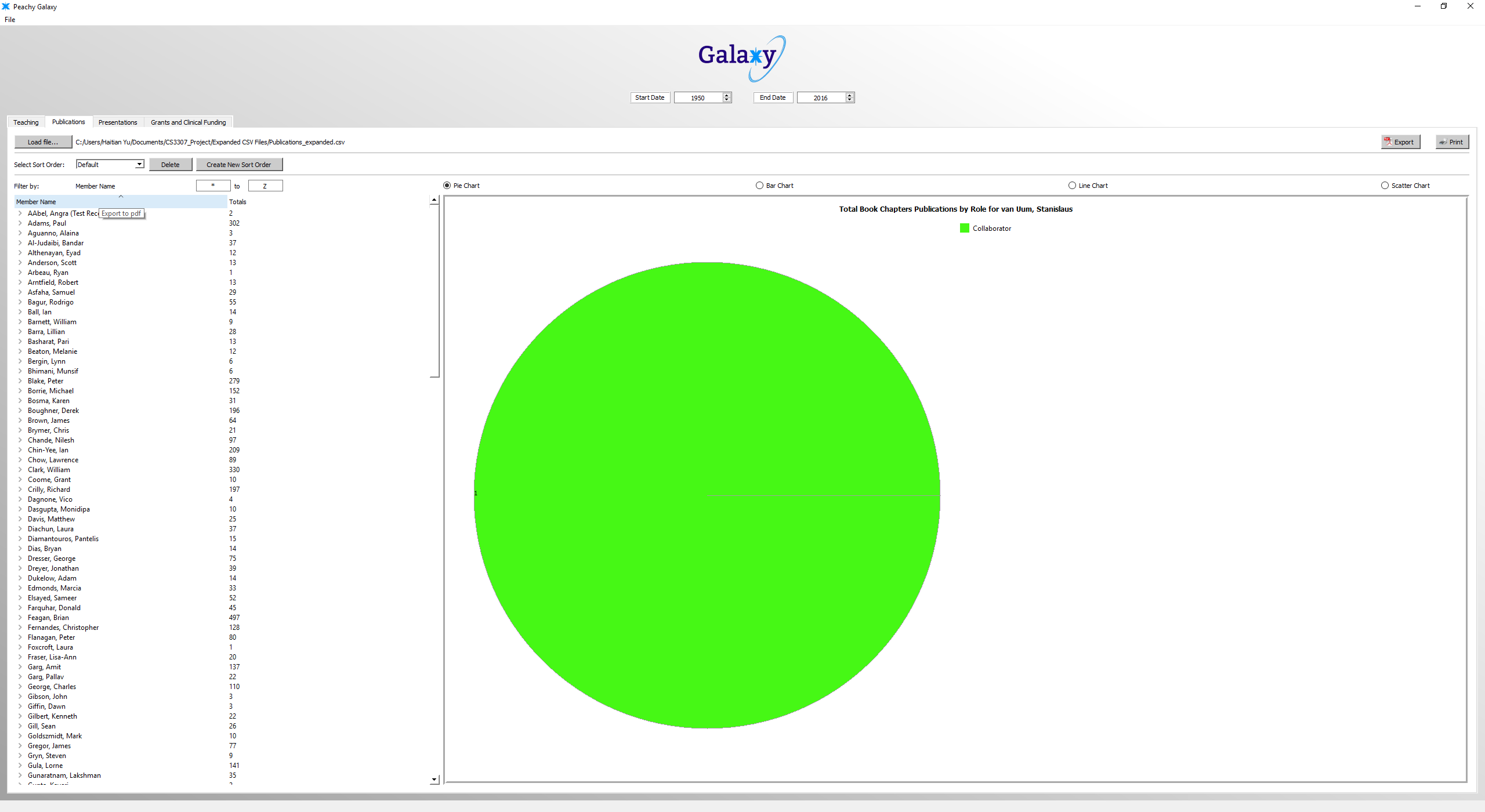
This is the description we got from Stephen, our TA:

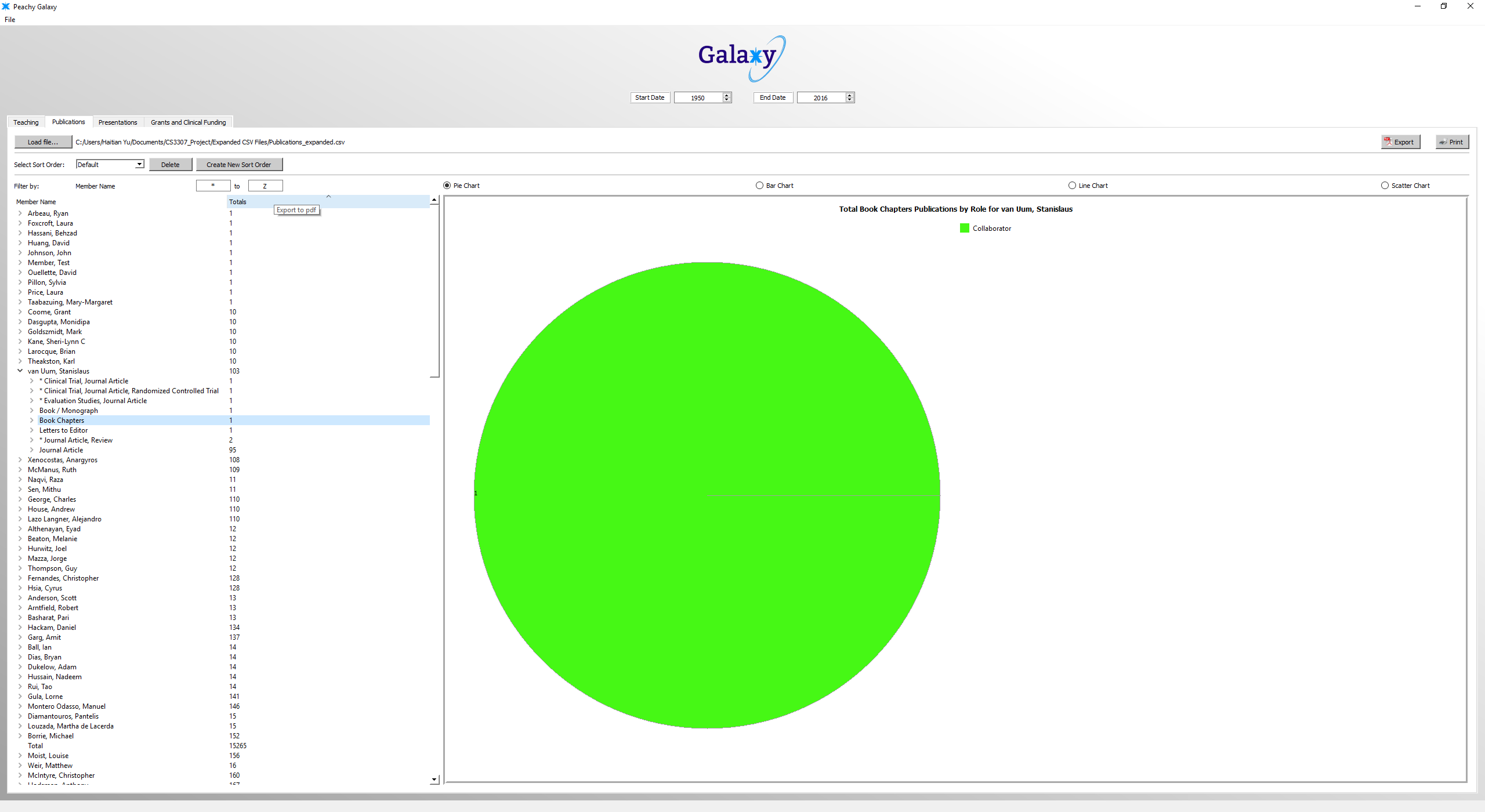
Indeed, sorting is already implemented, but the customer requires  
extensions to this functionality. Specifically:  
 - "the ability to sort by the member’s division" (mandatory req. #4)  
 - "the ability to sort by a user selected list" (mandatory req. #5) -  
**I believe this means sorting by clicking on a table header entry**

We have implemented the above to be able to sort by clicking on the table header.

**Origin:** Code created by Aboudi

**Supporting Example:**





I**D**: MandatoryRequirement6

**Description:** Fix all errors so finished product is error free

**Origin**: Coded by all members of the team

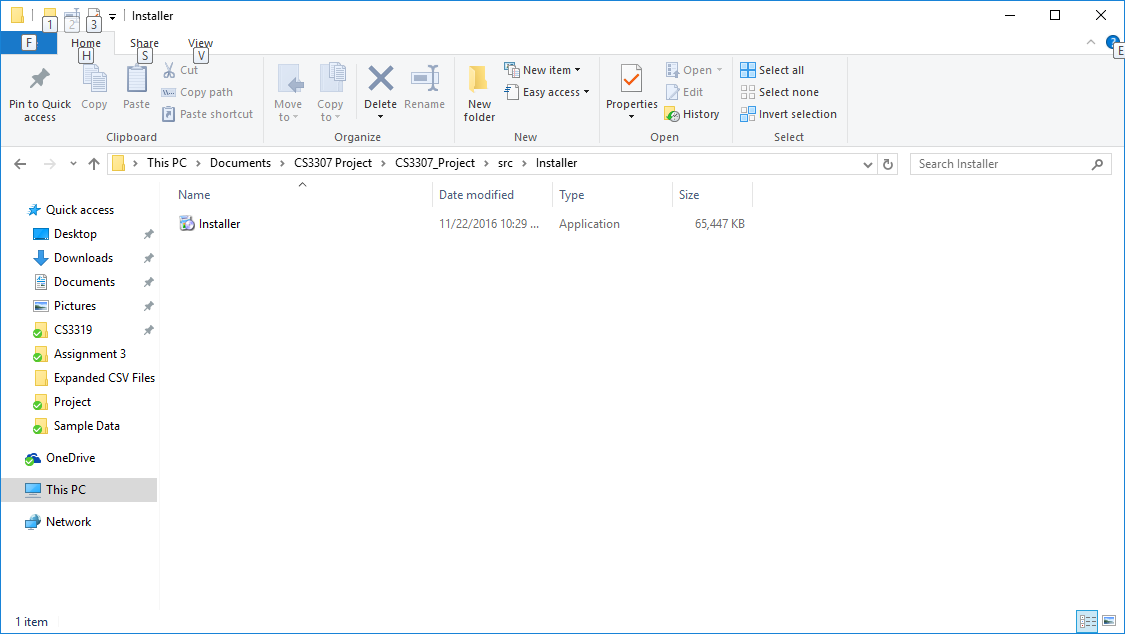
**Supporting Example**: **INCOMPLETE**

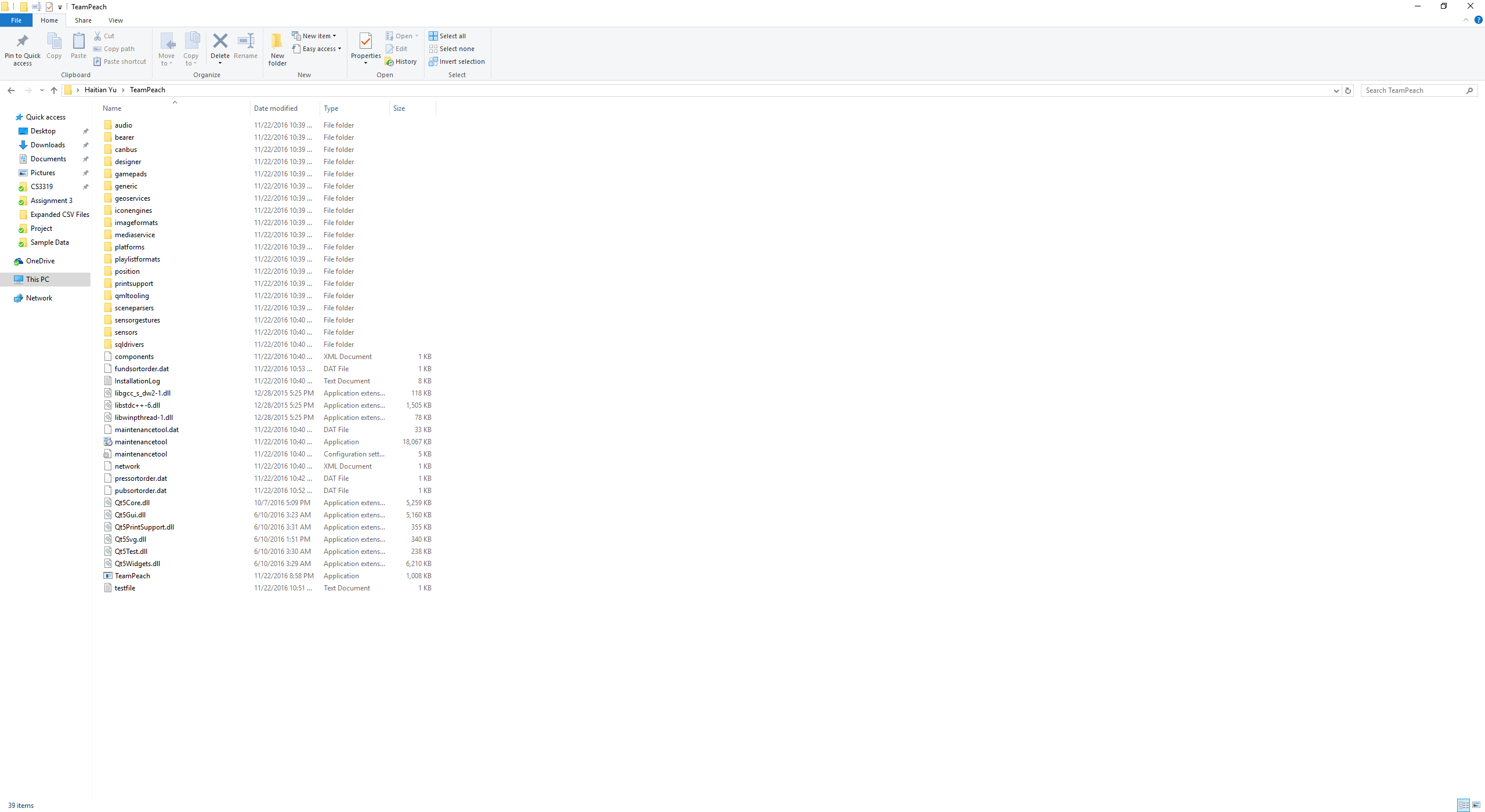
**ID:** MandatoryRequirement7

**Description:** Create a userproof install file that includes the executable file, supporting documentation, and all supporting files

**Origin:** Created by Anthony Giugno, Haitian Yu, and Claire Robichon

**Supporting Example:**





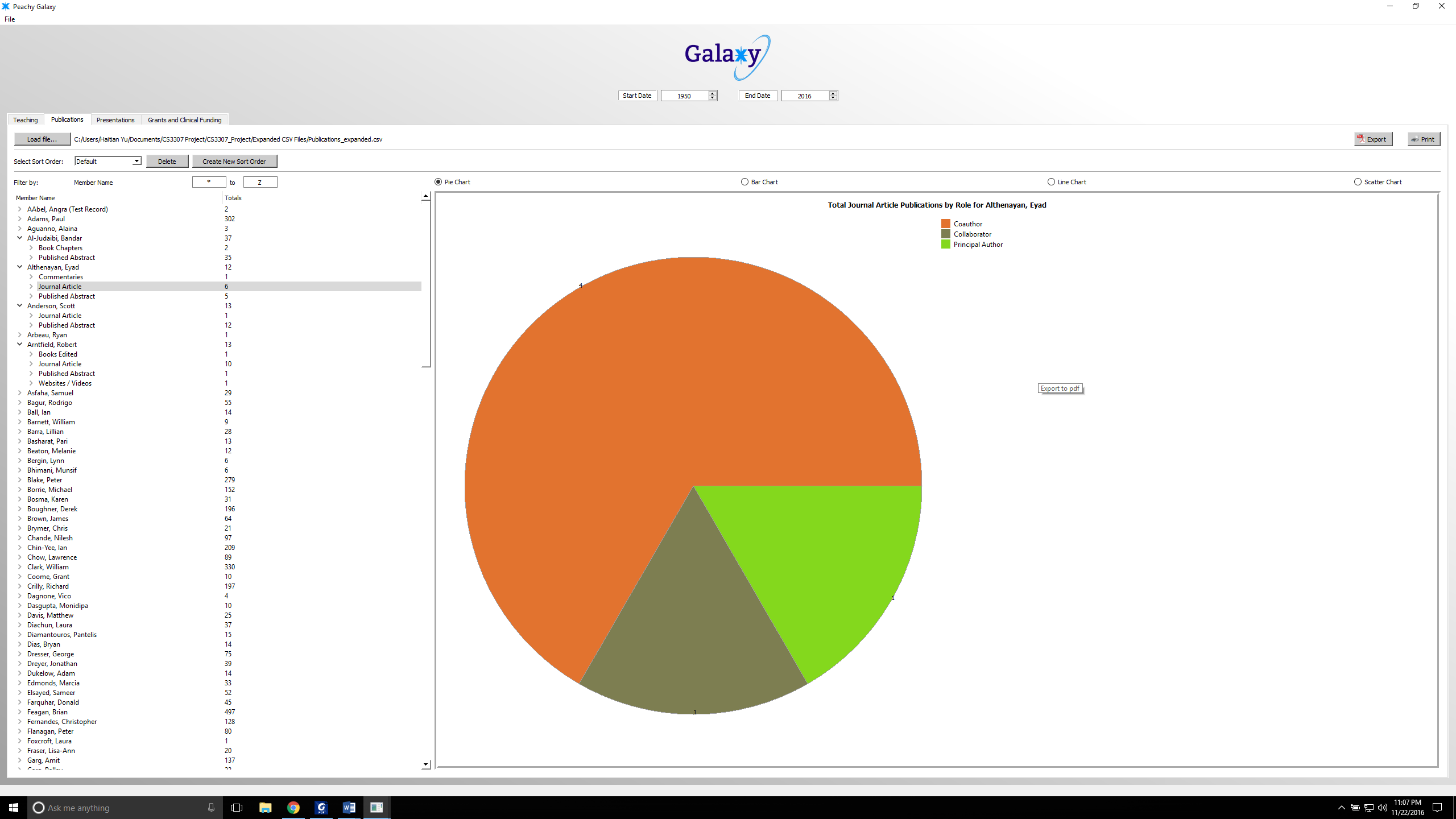
**ID:** MandatoryRequirement8

**Description**: verify the code deck for operation on Windows 7 or Windows 10

**Origin**: Created by Anthony Giugno, Haitian Yu, and Claire Robichon

**Supporting Example**:

The following print screen shows the application working on a Windows 10 operating system. It also works on the windows 7 operating system as well.



**Team Lynx – Test Cases**

Team peach specified 7 software requirements in their documentation for Peachy Galaxy:

* Loading data from file
* Error Processing
* Applying filters
* Using a custom sort order
* Visualizing data
* Printing to file
* Exporting to PDF

Team Lynx added the features below to the Peachy Galaxy:

* Saving application state
* Two new ways to visualize data (graphs)
* Column Sorting

Based on these requirements, we created test cases for various possible user test cases documented below. Test cases were developed in two methods: first, programmatically writing test cases using the QTest Library; second, manually using the GUI to replicate users actions. All programmatically written test cases can be found in the /test folder of the Peachy Galaxy project, and the ‘unit to test row’ in the following charts will provide the name of the method which was called in that test case.

Test cases that:

Testcase ID: 023, 021 (Both minor user experience issues)

|  |  |
| --- | --- |
| **Test Case ID:** | 001 |
| **Unit to test:** | QSORT\_LIST\_01 |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | QSortListIO will catch and throw exception when given an invalid location. |
| **Test Data:** | “NO FILE” |
| **Steps to be executed:** | Pass string to sorter, then call method .readList() and store  output as a QList |
| **Expected Result:** | QSortListIO success |
| **Actual Result:** | Debugger error |
| **Pass/Fail:** | Fail |
| **Comments:** | QSortListIO should contain a catch for invalid strings passed to the  class. |
| **Solution:** | Design a try/catch statments for invalid files locations. |

|  |  |
| --- | --- |
| **Test Case ID:** | 002 |
| **Unit to test:** | QSORT\_LIST\_02 |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | QSortListIO will catch and throw exception when given an invalid location. |
| **Test Data:** | NULL |
| **Steps to be executed:** | Pass string to sorter, then call method .readList() and store  output as a QList |
| **Expected Result:** | QSortListIO success |
| **Actual Result:** | Debugger error |
| **Pass/Fail:** | Fail |
| **Comments:** | QSortListIO should contain a catch for invalid strings passed to the  class. |
| **Solution:** | Design a try/catch statments for invalid files locations. |
|  |  |

|  |  |
| --- | --- |
| **Test Case ID:** | 003 |
| **Unit to test:** | QSORT\_LIST\_03 |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | QSortListIO will catch and throw exception when given an invalid location. |
| **Test Data:** | Grants\_expanded.csv |
| **Steps to be executed:** | Pass string to sorter, then call method .readList() and store  output as a QList |
| **Expected Result:** | QSortListIO success |
| **Actual Result:** | Success |
| **Pass/Fail:** | Pass |
| **Comments:** | QSortListIO should contain a catch for invalid strings passed to the  class. |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 004 |
| **Unit to test:** | QSORT\_LIST\_04 |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | QSortListIO will catch and throw exception when given an invalid location. |
| **Test Data:** | test\_file.txt |
| **Steps to be executed:** | Create text file, Pass string to sorter, then call method  .readList() and store output as a QList |
| **Expected Result:** | QSortListIO success |
| **Actual Result:** | Success |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 005 |
| **Unit to test:** | QCSV\_READER\_TEST\_09 |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | CSVReader will catch and throw exception when given an invalid  format |
| **Test Data:** | STAR 2015 Acuity STAR Western CS intro V2.pptx |
| **Steps to be executed:** | Pass string of location to CSVReader, then use method.  getData and pass it to vector, then check size to validate that it was accepted |
| **Expected Result:** | CVSReader catch invalid file |
| **Actual Result:** | Failed. CVSReader did not check file format |
| **Pass/Fail:** | Fail |
| **Comments:** | CSVReader should not be reliant upon the user the enter a .csv file.  There should be a try catch so the program does not crash |
| **Solution:** | Design a simple check to validate CSV files. |

|  |  |
| --- | --- |
| **Test Case ID:** | 006 |
| **Unit to test:** | Loading incorrect file from GUI |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | Error thrown on non CVS |
| **Test Data:** | Teaching\_test.docx |
| **Steps to be executed:** | 1. Select a tab 2. Click ‘load file’ 3. Pick an invalid CVS type 4. Wait for program response |
| **Expected Result:** | Program will not load invalid data |
| **Actual Result:** | Program did not load invalid file. |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 007 |
| **Unit to test:** | Loading incorrect tab types |
| **Requirements:** | 3.1.1 Final System Delivery (Loading data from file) |
| **Assumptions:** | Error thrown on wrong CSV tab type |
| **Test Data:** | Grants\_expanded.csv, GrantsClinicalFunding\_sample.csv,  Presentation\_expanded.csv, Presentation\_sample.csv, Program\_Teaching\_expanded.csv,  Publication\_expanded.csv, Publications\_sample.csv, Teaching\_sample.csv |
| **Steps to be executed:** | 1. Select a tab 2. Click ‘load file’ 3. Pick an invalid CVS type 4. Wait for program response |
| **Expected Result:** | Program will not load invalid tab type data |
| **Actual Result:** | Program did not load incorrect CVS file type. |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 008 |
| **Unit to test:** | Clicking Edit Button |
| **Requirements:** | 3.1.2 Final System Delivery (Error Proccessing) |
| **Assumptions:** | When clicking the Edit Button, Peachy galaxy should prompt me with the EditFields dialog. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Click edit button |
| **Expected Result:** | EditField dialog box will open |
| **Actual Result:** | EditField dialog box will open |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 009 |
| **Unit to test:** | Clicking Export Button |
| **Requirements:** | * 3.1.8 Final System Delivery (Exporting to PDF) |
| **Assumptions:** | When clicking the Export Button Peachy galaxy should prompt me with the “Export File” dialog. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Click export button |
| **Expected Result:** | “Export File” dialog box will open |
| **Actual Result:** | “Export File” dialog box will open |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 010 |
| **Unit to test:** | Clicking Print Button |
| **Requirements:** | 3.1.7 Final System Delivery (Printing to file) |
| **Assumptions:** | When clicking the Print Button Peachy galaxy should prompt me with the “Print” dialog. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Click print button |
| **Expected Result:** | “Print” dialog box will open |
| **Actual Result:** | “Print” dialog box will open |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 011 |
| **Unit to test:** | Clicking User Data |
| **Requirements:** | * 3.1.6 Final System Delivery (Visualizing data) |
| **Assumptions:** | When clicking a Member Name, their Data is visualized on the currently selected graph. |
| **Test Data:** | Program\_Teaching\_expanded.csv, Lavi, Shahar |
| **Steps to be executed:** | 1. Load file 2. Click Lavi, Shahar |
| **Expected Result:** | Data is modeled on Pie Chart |
| **Actual Result:** | Data is modeled on Pie Chart |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 012 |
| **Unit to test:** | Clicking Scatter Chart Button |
| **Requirements:** | * 3.1.6 Final System Delivery (Visualizing data) |
| **Assumptions:** | When a member name is selected and the Scatter Chart Button is selected, a scatter chart is produced |
| **Test Data:** | Program\_Teaching\_expanded.csv, Lavi, Shahar |
| **Steps to be executed:** | 1. Load file 2. Click Lavi, Shahar 3. Click Scatter Chart Button |
| **Expected Result:** | Data is modeled on Scatter Chart |
| **Actual Result:** | Data is modeled on Scatter Chart |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 013 |
| **Unit to test:** | Clicking Line Chart Button |
| **Requirements:** | * 3.1.6 Final System Delivery (Visualizing data) |
| **Assumptions:** | When a member name is selected and the Line Chart Button is selected, a line chart is produced |
| **Test Data:** | Program\_Teaching\_expanded.csv, Lavi, Shahar |
| **Steps to be executed:** | 1. Load file 2. Click Lavi, Shahar 3. Click Line Chart Button |
| **Expected Result:** | Data is modeled on Line Chart |
| **Actual Result:** | Data is modeled on Line Chart |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 014 |
| **Unit to test:** | Entering g to h for Filter by: Member Name |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for Filtering by: Member Name, all names between the characters are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | All entries between g and h are displayed |
| **Actual Result:** | All entries between g and h are displayed |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 015 |
| **Unit to test:** | Entering g to f for Filter by: Member Name |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for Filtering by: Member Name, all names between the characters are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | No entries are displayed |
| **Actual Result:** | No entries are displayed |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 016 |
| **Unit to test:** | Entering 1 to 9 for Filter by: Member Name |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for Filtering by: Member Name, all names between the characters are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | All entries are displayed |
| **Actual Result:** | All entries are displayed |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 017 |
| **Unit to test:** | Entering a to z for Filter by: Member Name |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for Filtering by: Member Name, all names between the characters are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | All entries are displayed |
| **Actual Result:** | Entries starting with lower case are displayed last |
| **Pass/Fail:** | Fail |
| **Comments:** | As the box parameters are not case sensitize, the program should display the names in alphabetical order regardless of case to avoid confusion. |
| **Solution:** | Convert names to lower case before filtering. |

|  |  |
| --- | --- |
| **Test Case ID:** | 018 |
| **Unit to test:** | Entering 1999 to 2015 for filtering by date |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for filtering by date, all names between the dates are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | All entries between the dates are displayed |
| **Actual Result:** | All entries between the dates are displayed |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 019 |
| **Unit to test:** | Entering 2015 to 2014 for filtering by date |
| **Requirements:** | * 3.1.4 Final System Delivery (Applying filters) |
| **Assumptions:** | When characters are entered into the fields for filtering by date, all names between the dates are displayed. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Enter g into first box and h into next box |
| **Expected Result:** | All entries between the dates are displayed, as the last date cannpt be entered |
| **Actual Result:** | All entries between the dates are displayed, as the last date cannpt be entered |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 020 |
| **Unit to test:** | Print Button |
| **Requirements:** | * 3.1.7 Print to File |
| **Assumptions:** | First print button will open printdialog, second print button will print the selected graph. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Select data 3. Press first print button 4. Press print in the dialog box 5. File prints |
| **Expected Result:** | File prints with desired information. |
| **Actual Result:** | File prints with desired information. |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 021 |
| **Unit to test:** | Print Button when no data is selected |
| **Requirements:** | * 3.1.4 Print to File |
| **Assumptions:** | Peachy galaxy will not let you print if you do not have any data selected. |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Press first print button 3. Press print in the dialog box 4. File prints |
| **Expected Result:** | File will not allow user to print |
| **Actual Result:** | File prints blank page. |
| **Pass/Fail:** | Fail |
| **Comments:** | Program should not allow user to print blank page |
| **Solution:** | If there is no data selected, disable to print button. |

|  |  |
| --- | --- |
| **Test Case ID:** | 022 |
| **Unit to test:** | Exporting data |
| **Requirements:** | * 3.1.8 Exporting to PDF |
| **Assumptions:** | Peachy galaxy will export your selected data |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Select data 3. Press export button 4. PDF saves |
| **Expected Result:** | File will export and pdf |
| **Actual Result:** | File exports as pdf |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 023 |
| **Unit to test:** | Exporting data |
| **Requirements:** | * 3.1.8 Exporting to PDF |
| **Assumptions:** | Peachy galaxy will not export to pdf if no data is selected |
| **Test Data:** | Program\_Teaching\_expanded.csv, |
| **Steps to be executed:** | 1. Load file 2. Press export button 3. Export data |
| **Expected Result:** | File will not allow user to export blank file |
| **Actual Result:** | File exports |
| **Pass/Fail:** | Fail |
| **Comments:** | Program should not allow user to export blank file |
| **Solution:** | If there is no data selected, disable the export button. |

|  |  |
| --- | --- |
| **Test Case ID:** | 024 |
| **Unit to test:** | Saving users state |
| **Requirements:** | * Saving users state |
| **Assumptions:** | Peachy galaxy will save users state when prompt accepted |
| **Test Data:** | Program\_Teaching\_expanded.csv |
| **Steps to be executed:** | 1. Load file 2. Select data 3. Exit program, and press yes to save state 4. Start program 5. Data reload into program |
| **Expected Result:** | File currently loaded will be reloaded upon restart |
| **Actual Result:** | File reloads into peachy galaxy |
| **Pass/Fail:** | Pass |
| **Comments:** | Potential bug: Save state does not save current tab, user must switch back to tab which data was loaded into |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 025 |
| **Unit to test:** | Saving users state |
| **Requirements:** | * Saving users state |
| **Assumptions:** | Peachy galaxy will save users only when accepted |
| **Test Data:** | Program\_Teaching\_expanded.csv |
| Steps **to be executed:** | 1. Load file 2. Select data 3. Exit program, and press yes to save state 4. Start program 5. Data reload into program 6. Load different data into program 7. Exit program and decline save 8. Start program, data from step 7 loads |
| **Expected Result:** | Save state will only save when prompt accepted |
| **Actual Result:** | Save state will only save when prompt accepted |
| **Pass/Fail:** | Pass |
| **Comments:** | Potential bug: Save state does not save current tab, user must switch back to tab which data was loaded into |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 026 |
| **Unit to test:** | Clicking Bar Button |
| **Requirements:** | * 3.1.6 Final System Delivery (Visualizing data) |
| **Assumptions:** | When a member name is selected and the Bar Chart Button is selected, a bar chart is produced |
| **Test Data:** | Program\_Teaching\_expanded.csv, Lavi, Shahar |
| **Steps to be executed:** | 1. Load file 2. Click Lavi, Shahar 3. Click Bar Chart Button |
| **Expected Result:** | Data is modeled on Bar Chart |
| **Actual Result:** | Data is modeled on Bar Chart |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 027 |
| **Unit to test:** | Clicking PieChart Button |
| **Requirements:** | * 3.1.6 Final System Delivery (Visualizing data) |
| **Assumptions:** | When a member name is selected and the Pie chart Button is selected, a bar chart is produced |
| **Test Data:** | Program\_Teaching\_expanded.csv, Lavi, Shahar |
| **Steps to be executed:** | 1. Load file 2. Click Lavi, Shahar 3. Click Pie chart Button |
| **Expected Result:** | Data is modeled on Pie Chart |
| **Actual Result:** | Data is modeled on Pie Chart |
| **Pass/Fail:** | Pass |
| **Comments:** | N/A |
| **Solution:** | N/A |

|  |  |
| --- | --- |
| **Test Case ID:** | 028 |
| **Unit to test:** | Column Sorting |
| **Requirements:** | Column Sorting |
| **Assumptions:** | Loaded data can be sorted by their column name. |
| **Test Data:** | Program\_Teaching\_expanded.csv |
| **Steps to be executed:** | 1. Load file 2. Click on column name |
| **Expected Result:** | Data is sorted automatically |
| **Actual Result:** | Data is sorted automatically |
| **Pass/Fail:** | Pass |
| **Comments:** | Column is sorted lexicographically instead of incrementally |
| **Solution:** | N/A |

**Test Matrix**



**Diagrams**

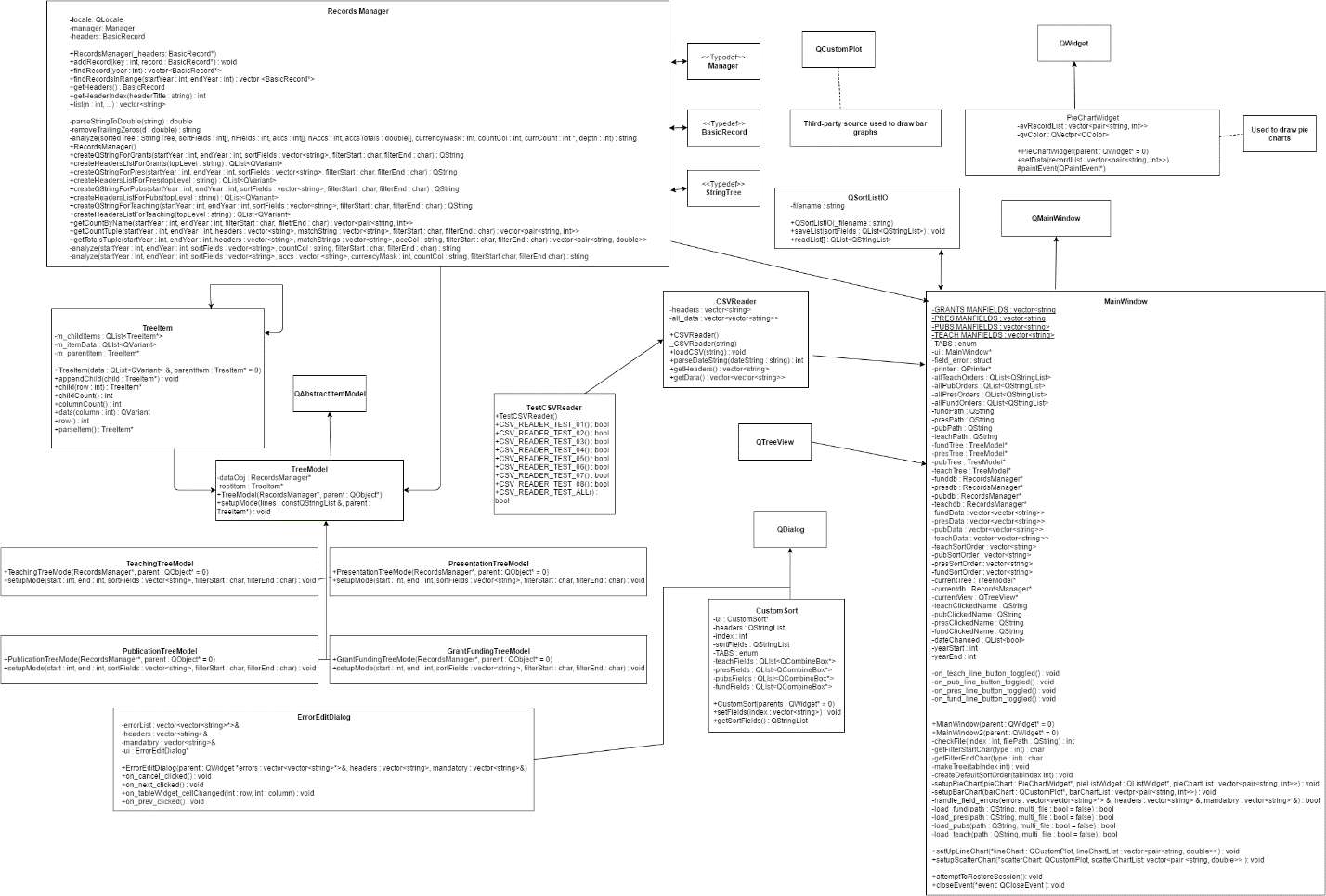
**Case Diagram Description:**

Above is an updated version of the Peachy Galaxy use case diagram for team Lynx. A user is anyone who runs the program, namely managers and faculty. One main use case is “Navigate Dashboard”, and the other is “Visualize Data”. “Navigate Dashboard” refers to the user’s interaction with the system UI. Through this, they are able to sort data and apply filters. “Visualize Data” refers to the various ways the system can output the data utilized. The user has the options sort the data and apply filters, as well as print the file and export data to a PDF. Both of these uses also utilize “Load Data Files”, which represents the user opening CSV files and processing their data through the system. This is also where the system processes errors. Using new methods, the user may also navigate through the various errors encountered.



**Class Diagram Description**

The system utilizes the original classes from team Peach. Several methods have been added to modify the system. The methods “on\_cancel\_clicked”, “on\_next\_clicked”, “on\_tableWidget\_cellChanged”, andand “on\_prev\_clicked” have been added to “ErrorEditDialog”. class. These methods were added to the class to allow the user to navigate through errors identified in the system and fill in missing data, in accordance with the customer requests. Additionally, “setupLineChart” was added to the “MainWindow” class. This method allows the user to visualize the input data in the form of a line graph, in accordance with the customer’s request to have another way to visualize the data. This method also makes use of “on\_pub\_line\_button\_toggled”, “on\_pres\_line\_button\_toggled”, “on\_teach\_line\_button\_toggled”, and “on\_fund\_line\_button\_toggled”, to allow the user to choose which data is visualized. The method “attemptToRestoreSession” is used to retrieve a previous session, which would have been saved during a “closeEvent”, activated when the user ends a session, to store data from that session. “setUpLineChart” and “setupScatterChart” are both used to produce alternative graph types as requested by the user.



**User Clicks Load Button Sequence Diagram**

This diagram remains the same as the one used by team Peach. It illustrates the scenario, initiated by a user, to load a file, and the steps taken to do so. The initial action is the user clicks the “load Data Button” from the “MainWindow”. This button exists in four instances, one for each type of data. The process remains the same for each type, but uses the versions of each method for each respective type. The system then loads the file data and checks it to match the required type using the method “checkFile”. The final steps are handled by a “RecordsManager”, created by the main window. The system then retrieves the data and sorts it by adding it to a “TreeModel” of the associated data type. The “analyze” methods is then called to order the data, which is then output back to the “MainWindow”.



**User Fills In Data Sequence Diagram**

This action is initiated from the “MainWindow” when the user Loads a data set. Before it is added to the “MainWindow” The program checks for missing entries that are required. If missing entries are found, a “QMessageBox” is created requesting if the user would like to edit the entries. If the user selects “Yes”, the following actions occur. First a new “ErrorEditDialogBox” is created and the “MainWindow” passes on the data set. The “ErrorEditDialogBox” then cleans the allocated memory. A “QWidget” is then generated to hold a “QWidgetTableItem”, which are populated from the passed data set. All potential entries are then editable from the “ErrorEditDialogBox”. From here the user can use “on\_nextclicks” and “on\_prev\_clicked” to navigate through the widget items and edit the data. As the data is navigated, the “on\_tableWidget\_cellChanges” update the widget items. The user can then use “on\_save\_clicked” to save the data and return to the main window, or”on\_cancel\_clicked” to navigate back to the “MainWindow” without changing the data.



**User Creates New Sort Order Sequence Diagram**

The action is initiated from the “MainWindow” when a user clicks the button to create a new sort order. This begins an event to create a new sort order, each specific to the data type for the open window. A “CustomSort” object is then created. This then creates new “QWidget” from which the user can input their desired sort parameters and name their new sort type. The fields within the widget are then set, after which a “QStringList” object is created and populated with the fields. The “QStringList” is then transferred back to be stored inside a variable of the “MainWindow” using the “addItem” method. A “QSortListIO” object is then created so that the new sort may be saved to the “MainWindow” using the “saveList” method. Lastly, the “CustomSort” object is deleted.



**Package Diagram**

The package diagram remains the same as team Peach’s package diagram. There is still only one type of relationship, dependency, denoted by the solid arrows. Each arrow begins at the source and ends at the target.

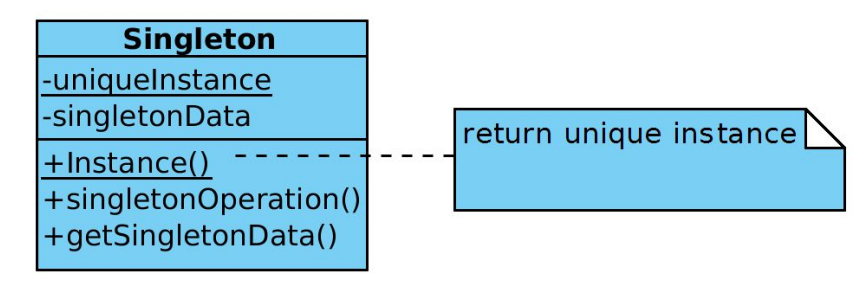


**Design Patterns:**

For most of our deliverables, we followed the design patterns of Peachy Galaxy as we are adding additional functionality and not creating the program from scratch. Therefore, all our deliverables are coded as additional functions in Peachy Galaxy’s classes.

**Singleton**

Singleton design solves the following problem: application needs one, and only one, instance of an object. The class is responsible for keeping track of the single instance (prevents other instances from being created as well as controls access to the instance).



**Benefits to Singleton:**

* Using a singleton gave us an advantage over global variables because we can be 100% sure about the number of instances. It also reduces namespace by avoiding global variables that store single instances
* Easier to refine operation and representation as the Singleton class can be subclassed
* Singleton encapsulates the instance; therefore, it has control over access and creation of instances
* Flexible number of instances (if we want to add instances)

**Implementation in C++:**

Singleton design is used in the implementation of MainWindow (controls the graphical user interface). The additions we have made to this class are the new graph functions setupScatterChart and setupLineChart which continue to follow the singleton design pattern.

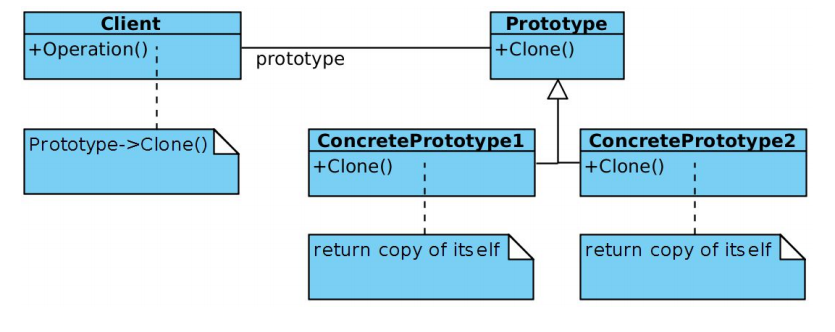
**Sources:**

<https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm>

<https://sourcemaking.com/design_patterns/singleton>

**Prototype**

Prototype design solves the following problem: application “hard wires” the class of object to create in each “new” expression. If an application/program has many objects which have similar behavior (slight differences in behavior), we can use prototype design to clone instances of the desired class.



**Benefits to Prototype:**

* Hides the concrete product classes from client (reduces the number of names clients know about)
* Allows client to work with application specific classes without modification
* Allows for adding and removing products at run-time: can introduce new product class into system by registering a prototype instance.
* Reduce number of needed classes: a client can exhibit new behavior by delegating responsibility to a prototype. By changing values for object variables, we can basically define “new” classes without doing so
* Prototyping, unlike Factory Method, doesn’t require sub-classing

**Implementation in C++:**

Prototype design is used in the implementation of the TreeModel class. It uses a prototype manager which means when the number of prototypes in a system isn’t fixed, it keeps a registry of available prototypes.

Sources:

<https://sourcemaking.com/design_patterns/prototype>

<https://www.tutorialspoint.com/design_pattern/prototype_pattern.htm>

**Agent Task View**



**Timeline + Achievements**





**Plan for Final Deliverable**

