

Team Orion
Upgrade of the Peachy-Galaxy
Final Submission Report
(12-07-2017)



Course: 3307a (Object-Oriented Programming)

Instructor: Nazim H. Madhavji

Teaching Assistant: Daniel Robert Page

Project Manager: Paul Henderson

Lead Programmer: Chris Brown

Lead Designer: James Walsh

Chief of Quality Assurance/Documentation: Ming Yang

Designer: Yuchen Wang

Developer: Jeffrey Hsu

Identification Tags	4
Requirements Implemented	5
Mandatory Requirement 2	5
Mandatory Requirement 3	9
Mandatory Requirement 4	13
Mandatory Requirement 5	17
Mandatory Requirement 9	20
Stretch Goal 2 - Change / Modify Sort Order	23
Stretch Goal 3 - Training Video	26
Test Cases for Requirements	27
Test ID U:	34
Testing Matrix	36
System Design	38
Use Case Diagram v3	38
Class Diagram v3	39
Sequence Diagram v2	41
Sequence Diagram v2-#2 - Error Editing	42
Sequence Diagram #3 - Visualization with Graphs	43
Package Diagram v2	44
Design Patterns	45
Singleton	45
Prototype	46
Code and Design Inspection	48
Implementation	60
Development Plans	61
Stage 1 Timeline	61
Stage 2 Timeline	62
Final Stage Timeline	62
Agent-Task View	63
Stage 1 Task Allocations	63
Stage 2 Task Allocation	63
Final Stage Task Allocation	64
Lessons Learnt	65
Top Three Lesson Learnt (Technical issues)	65

Retrospective Analysis (Human issues)	66
Value	67

66
67

Identification Tags

I.D.	Objective
M.R.1	Mandatory Requirement 1, the assessment of the Phase 1 Project
M.R.2-1	Mandatory Requirement 2, this is the Line graph for the system
M.R.2-2	Second Graph for Mandatory Requirement 2
M.R.3	Mandatory Requirement 3 - Ability to save Session
M.R.4	Mandatory Requirement 4 - Added "Division" as sort category
M.R.5	Mandatory Requirement 5 - Ability to sort by a user selected list
M.R.6	Mandatory Requirement 6 - Correct all errors
M.R.7	Mandatory Requirement 7 - Deliver user-proof install file
M.R.8	Mandatory Requirement 8 - Make sure program works on Windows 7
M.R.9	Mandatory Requirement 9 - User can go through erroneous entries to fix
S.R.2	Stretch Requirement 2 - Change/Modify Sort Order
S.R.3	Stretch Requirement 3 - Supply a training video demonstrating installation and use of your program
B1	(Bug 1) - New CSV files that will be used for the program cause crashes

Requirements Implemented

Mandatory Requirement 2

ID M.R.2 (M.R.2-1 and M.R.2-2)

Context

Add two new graph types to the reporting options Extension of existing graphs at sub-categories
Add 2 New Types: Line, plus one of your choice Graph / Chart entire department, division or list.

Description

We implemented a line chart representation of data to the reporting options (Mandatory Requirement). In Phase I of the project, Team Peach implemented two different kinds of visualization for the data: a pie chart and a bar chart. In Phase II, we have added two other options for users: a line chart and a scatter plot. We added GUI elements in the same style as Phase I uses allowing the user to check another box to change their view. This required learning how to make user of the QT Designer to add widgets and bind functionality to the 'slots' of GUI elements. We added a method to the `MainWindow` class that parses the list representation of the data to render it in the correct visual form.

Originally, our release of this feature had a bug where it would only display user data for the last two years (2015 and 2016). We have reevaluated our parsing of the data files so that any field that includes year data will be properly presented on the time series.

Additionally, in the Phase I release of the code, the software would only draw pie and bar charts when a single name was selected in the left hand table. We have implemented the ability to view multiple users data by clicking on the 'Total' entry present at the bottom of every view. The current view can of course be modified by changing the sort order. This allows users of the software to compare different members of the faculty.

The final chart type was the scatter plot, which is available in the Grants and Funding tab of the program. When the user selects 'Total' from the list of names, they are able to observe patterns between the number of fundings a faculty member earns and the total amount of money they get. The logarithm is taken from this data because the differences in money funded are often powers of ten and would thus be difficult to make sense of on a single linear chart.

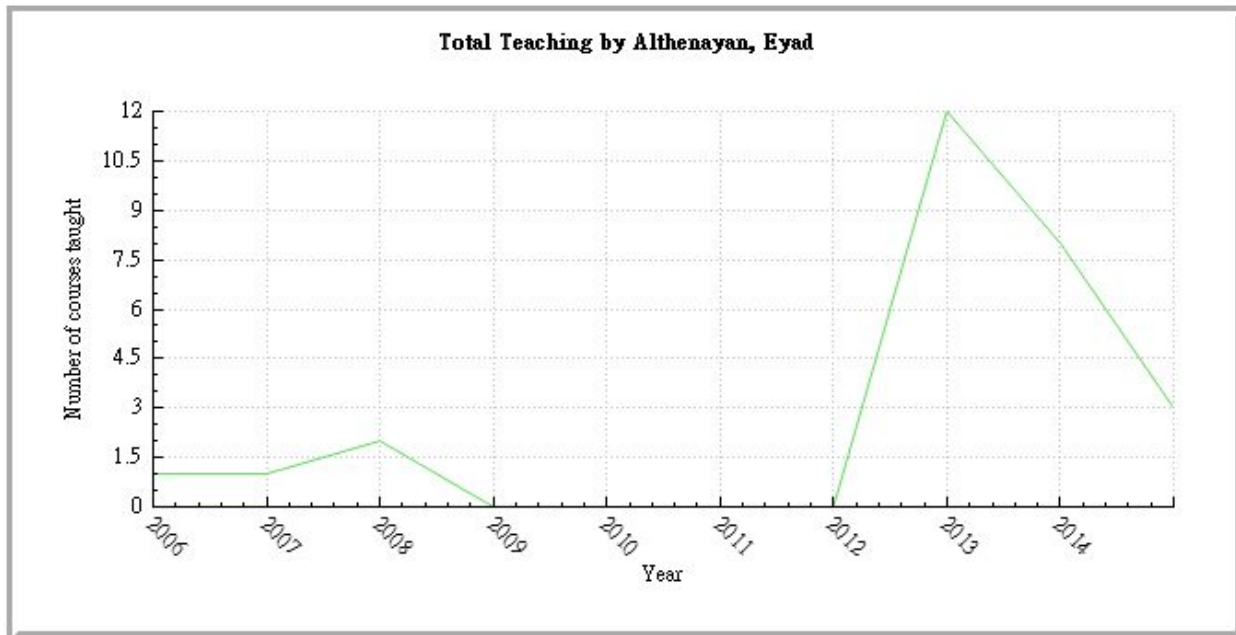
Origin

This was a mandatory requirement outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Examples

A demonstration of the Line Chart time series. Note that the time series features is only available for visualization when the current view includes 'Date' as a sort order. This is done by creating a new 'Custom Sort Order' by clicking the button labelled thus. The time series works the same way for all types of CSV files loaded.

☐ Pie Chart ☐ Bar Chart ☒ Line Chart

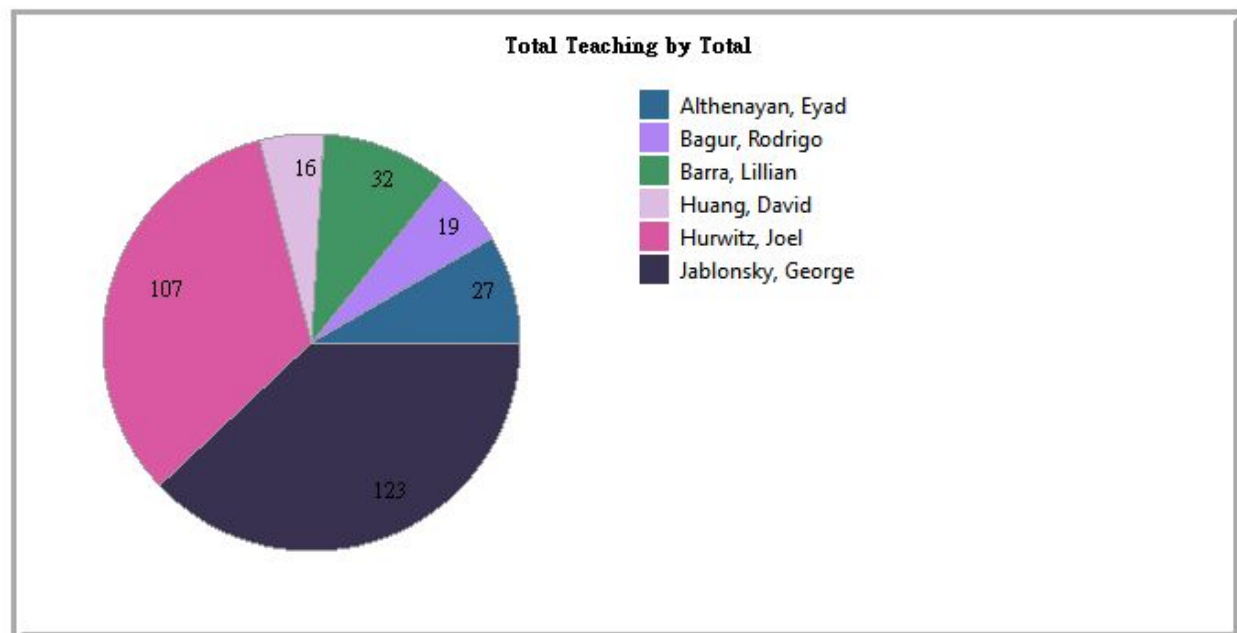


The following two charts represent the addition of the ability to view multiple people's data at once. After sorting what information is present, you can click on the 'Total' entry at the bottom of the column to view pie and bar charts representative of the entire list.

☒ Pie Chart

☐ Bar Chart

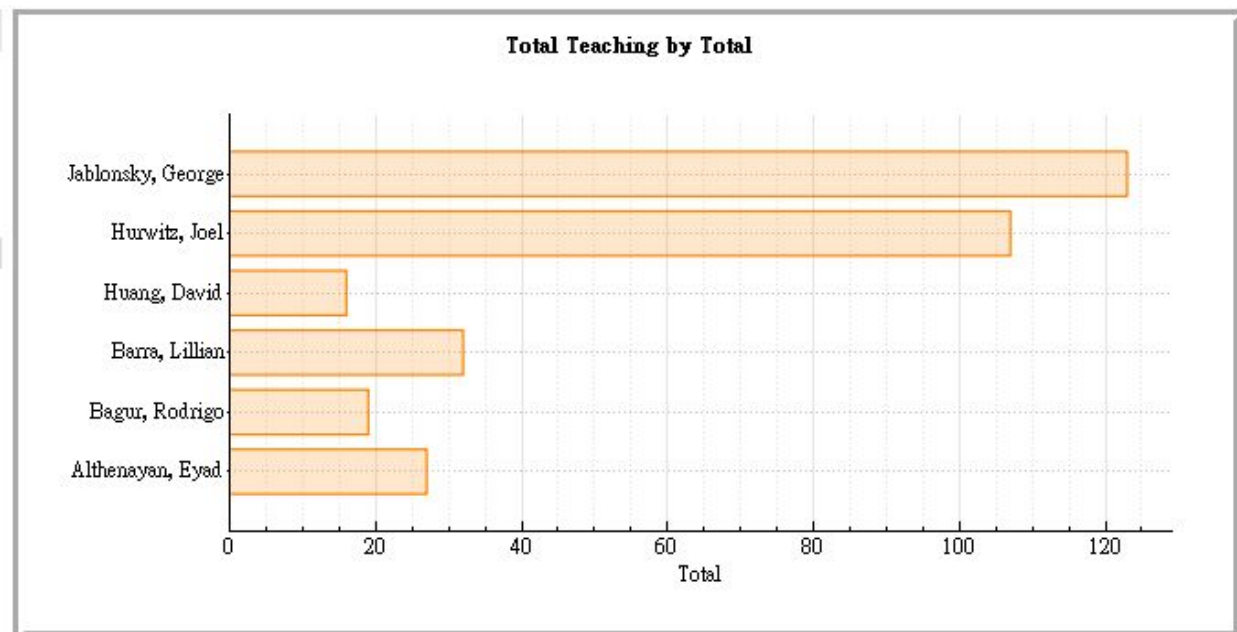
☐ Line Chart



☐ Pie Chart

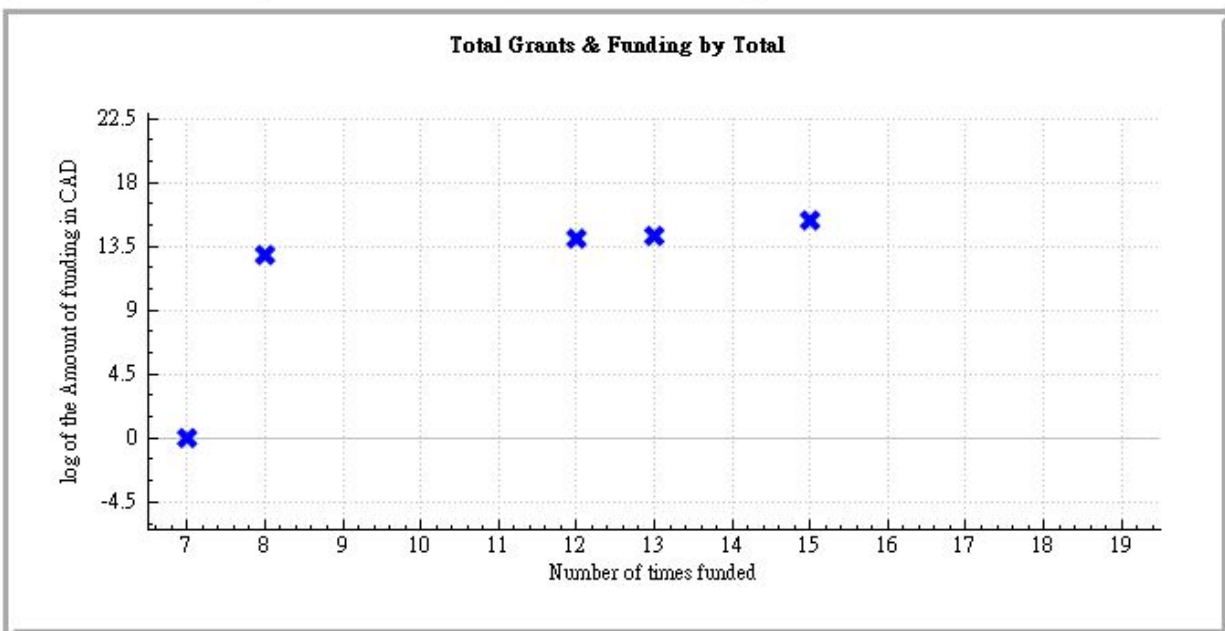
☒ Bar Chart

☐ Line Chart



The final chart represents the fourth type of graph added to the Peach Galaxy program. This feature is accessible for Grant type CSV files and allows users to observe relationships between the total number of grants professors have received and the amount of funding in dollars that they have received. When the differences between the amounts of funding are great, the values are compared based on the logarithm instead.

☐ Pie Chart ☐ Bar Chart ☐ Line Chart ☒ Scatter Plot



Mandatory Requirement 3

ID M.R.3

Context

The ability to save the session state

Description

One of the requirements specified by the client is for the program to have the ability to save the current session state so the user can resume what they were last doing with it. Currently we have implemented methods that remember which files have been loaded during the previous session. We were able to retool the class QSortListIO from Phase 1 into QFileIO which accepts a single string containing the path to a file that was loaded instead of a list of strings containing methods to sort its data. QFileIO takes a QString of the file path of each type of .csv loaded during a session and serializes it to a .dat file in the folder the program is running from. The mainwindow class searches for .dat files and loads the path stored inside if such files are found. In the event that no .dat file is located for a field (teaching, publications, etc.), the program assumes that there was no such file loaded during the previous session and leaves that field empty. Once the program locates an existing file it is loaded automatically and declines the error handling prompt which requests the user to fill in missing mandatory fields. For all files loaded manually, the prompt will still appear as usual.

Origin

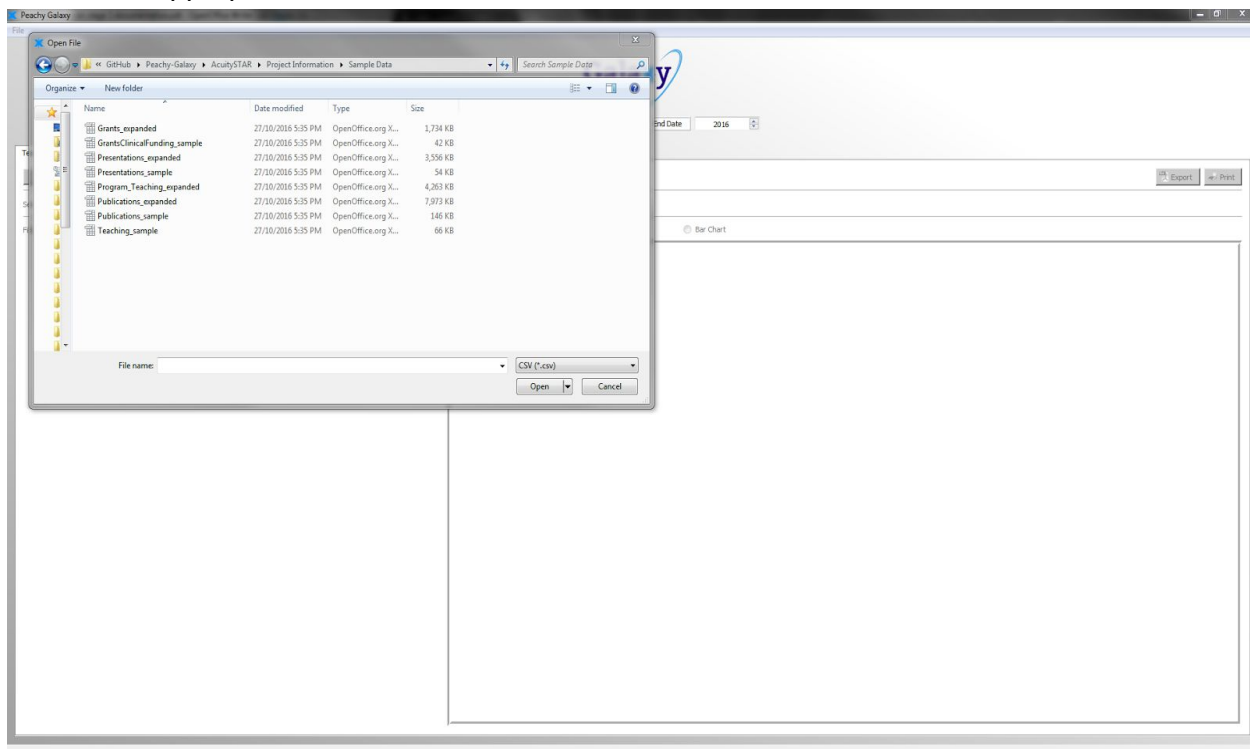
This was a mandatory requirement outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Example

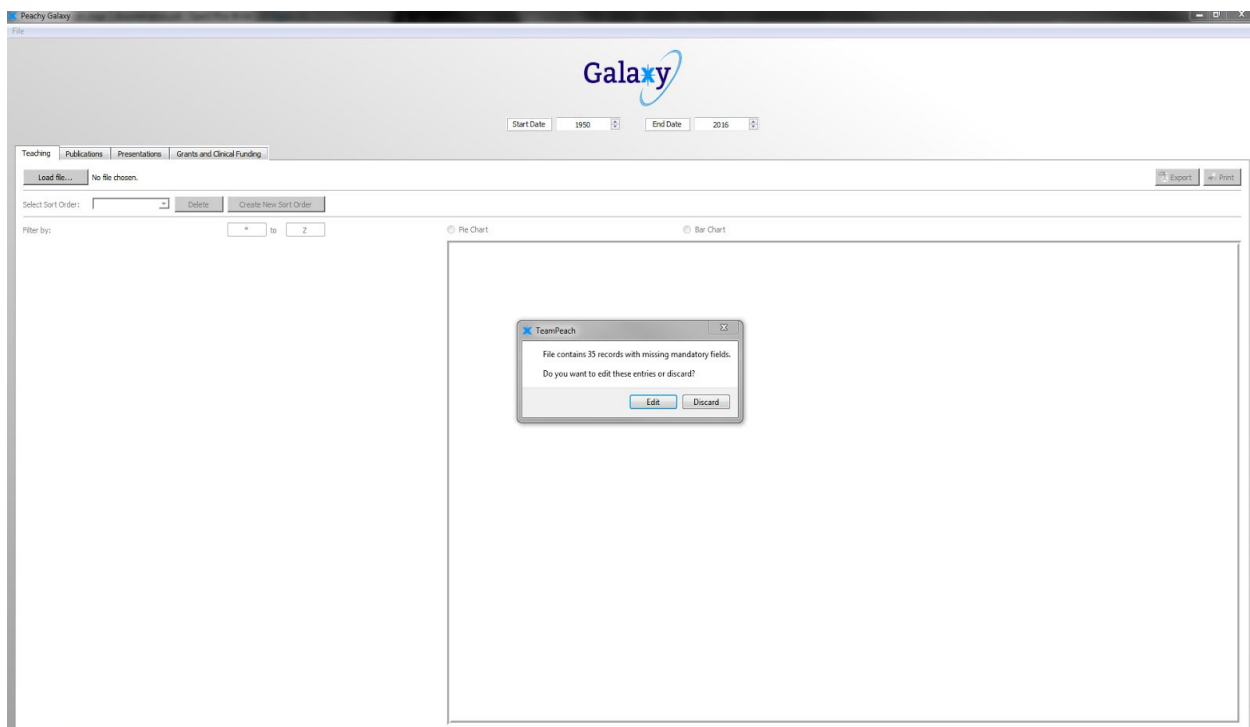
1. Open program normally



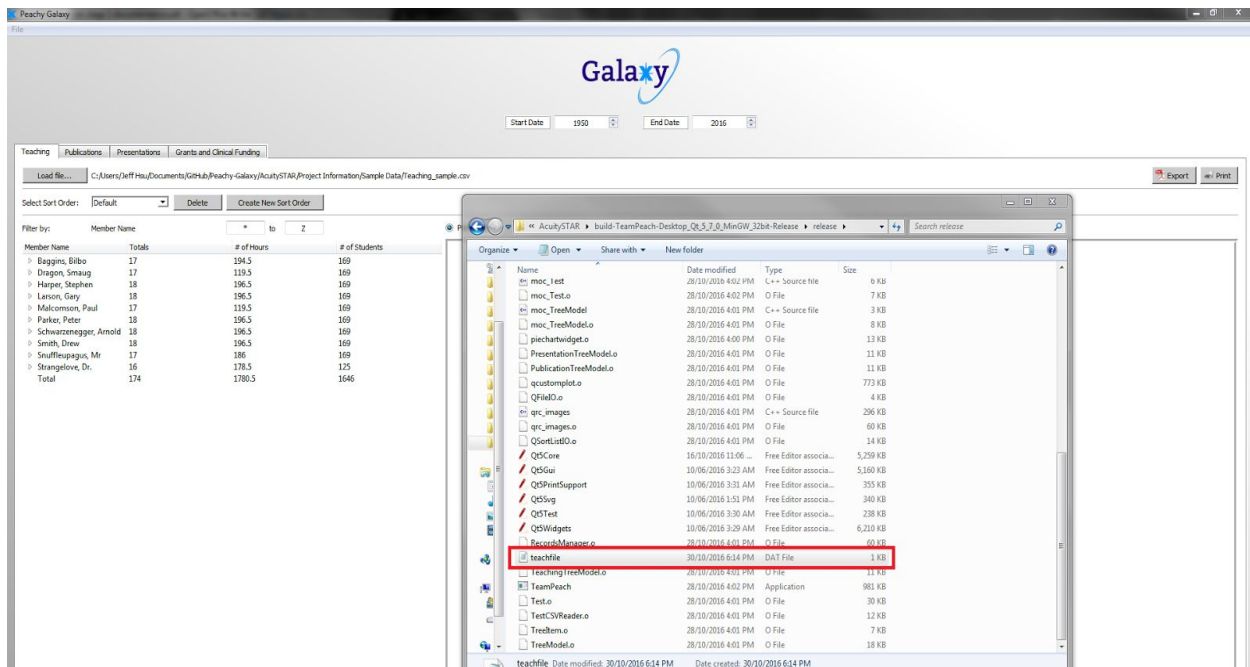
2. Load an appropriate file



3. Loading files manually will cause this prompt to appear if there are missing mandatory fields



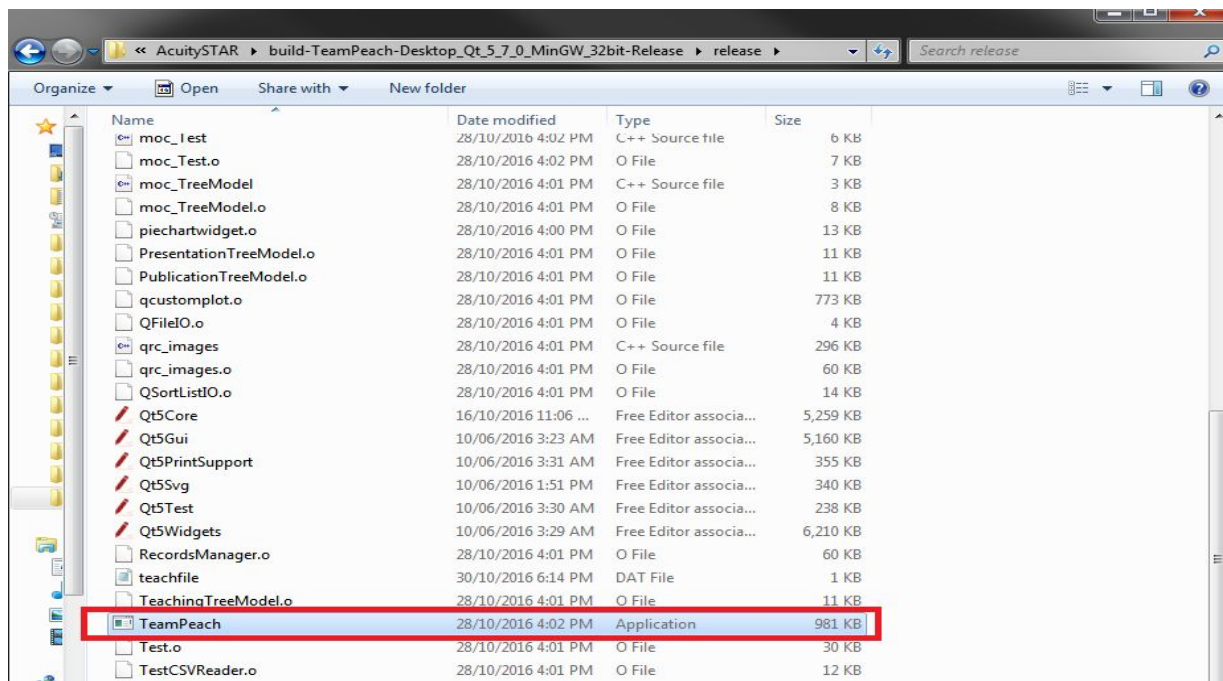
4. The file is loaded and the path to it is saved as a .dat file



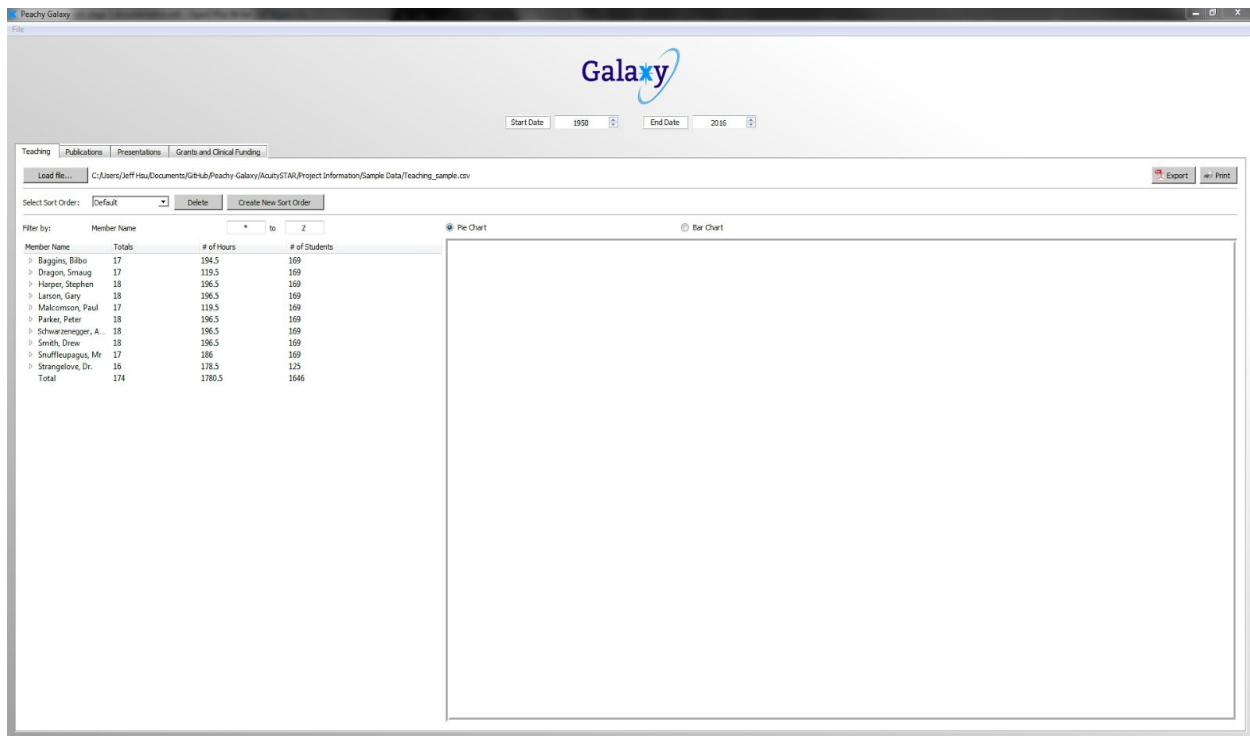
Contents of the .dat file



5. Close and reopen the program



6. You will see that on startup the file opened during the previous session will be loaded automatically.



Mandatory Requirement 4

ID M.R.4

Context

Expand Phase I Project by adding the ability to sort by the member's division.

Description

We expanded Phase I Project by adding the ability to sort by the member's division. In Phase I Project, the users can not be sorted by Division. In Phase II, we have added the ability to sort data by a member's division in Teaching, Presentations, Publications and Funding. They can create a new sort order, and only sort data by the member's division.

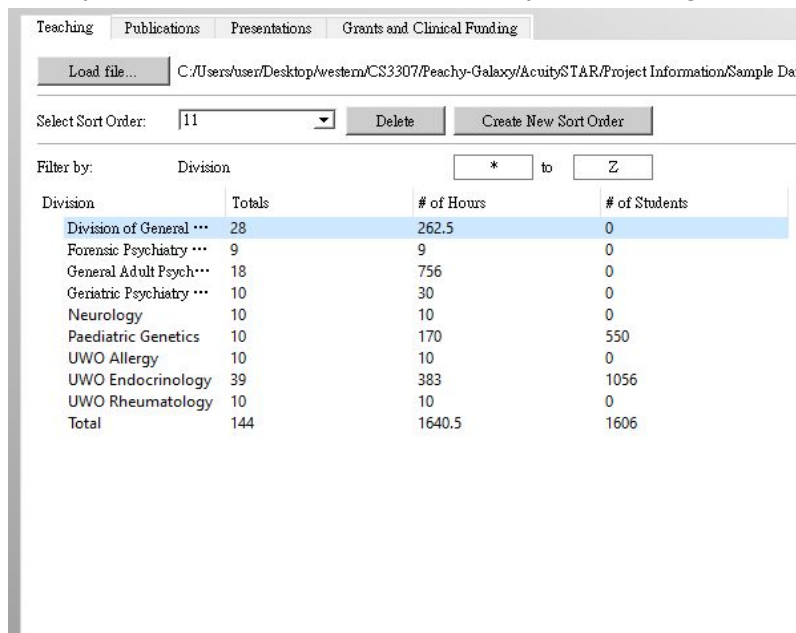
We added a new variable and a judgement statement in the "checkFile" function in the mainwindow.cpp file. So the users are able to sort data by the member's division if they choose division in the Create New Sort Order (CustomSort.cpp) dialog widget.

Origin

This was a mandatory requirement outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Example

Sort by the member's division individually in Teaching



Division	Totals	# of Hours	# of Students
Division of General ...	28	262.5	0
Forensic Psychiatry ...	9	9	0
General Adult Psych...	18	756	0
Geriatric Psychiatry ...	10	30	0
Neurology	10	10	0
Paediatric Genetics	10	170	550
UWO Allergy	10	10	0
UWO Endocrinology	39	383	1056
UWO Rheumatology	10	10	0
Total	144	1640.5	1606

Sort by the member's division individually in Publications

Teaching Publications Presentations Grants and Clinical Funding

Load file... C:/Users/user/Desktop/western/CS3307/Peachy-Galaxy/AcuitySTAR/Project Information/Sample Data/Publicatio

Select Sort Order: [1111] Delete Create New Sort Order Edit Sort Order Custom List

Filter by: Division * to Z

Division	Totals
Allergy	110
Cardiology	5381
Clinical Pharmacology	556
Critical Care	428
Emergency Medicine	607
Endocrinology	2078
Gastroenterology	1031
General Internal Medicine	757
Geriatrics	839
Hematology	1225
Infectious Diseases	215
Nephrology	2904
Respirology	1125
Rheumatology	1004
Total	18260

Sort by the member's division individually in Presentations

Teaching Publications Presentations Grants and Clinical Funding

Load file... C:/Users/user/Desktop/western/CS3307/Peachy-Galaxy/AcuitySTAR/Project Information/Sample Data/Presentations_

Select Sort Order: [see] Delete Create New Sort Order Edit Sort Order Custom List

Filter by: Division * to Z

Division	# Presentations
Allergy	12
Cardiology	1601
Clinical Pharmacology	403
Critical Care	434
Emergency Medicine	753
Endocrinology	1302
Gastroenterology	406
General Internal Medicine	443
Geriatrics	530
Hematology	331
Infectious Diseases	216
Nephrology	1575
Respirology	723
Rheumatology	187
Total	8916

Sort by the member's division individually in Funding

Teaching Publications Presentations Grants and Clinical Funding

Load file... C:/Users/user/Desktop/western/CS3307/Peachy-Galaxy/AcuitySTAR/Project Information/Sample Data/Grants_exp

Select Sort Order: Delete Create New Sort Order Edit Sort Order Custom List

Filter by: Division * to Z

Division	Total	Total \$
Allergy	70	\$0.00
Cardiology	1073	\$201,301,013.96
Clinical Pharmacology	92	\$40,840,829.00
Critical Care	162	\$45,670,726.00
Emergency Medicine	64	\$14,304,408.05
Endocrinology	416	\$160,258,729.31
Gastroenterology	247	\$61,839,763.05
General Internal Medicine	207	\$86,278,120.00
Geriatrics	250	\$99,981,762.05
Hematology	284	\$60,844,651.94
Infectious Diseases	44	\$2,086,488.32
Nephrology	616	\$207,949,756.12
Respirology	331	\$41,111,706.85
Rheumatology	171	\$24,485,147.28
Total	4027	\$1,046,953,101.93

Create new sort order, sort by division, start date, then program

Create New Sort Order

New Custom Sort Order

Enter name of new custom sort:

66

Select order:

Division

Start Date

Program

Save Cancel

Teaching

Publications

Presentations

Grants and Clinical Funding

Load file...

C:/Users/user/Desktop/western/CS3307/Peachy-Galaxy/AcuitySTAR/Project Information/Sample :

Select Sort Order:

66

Delete

Create New Sort Order

Filter by:

Division

*

to

Z

Division	Totals	# of Hours	# of Students
▼ Division of General ...	28	262.5	0
▼ 1999	9	94.5	0
Postgraduat...	9	94.5	0
> 2010	10	150	0
> 2015	9	18	0
▼ Forensic Psychiatry ...	9	9	0
> 2015	9	9	0
▼ General Adult Psych...	18	756	0
▼ 2010	8	616	0
Undergradu...	8	616	0
> 2015	10	140	0
> Geriatric Psychiatry ...	10	30	0
> Neurology	10	10	0
> Paediatric Genetics	10	170	550
> UWO Allergy	10	10	0
> UWO Endocrinology	39	383	1056
> UWO Rheumatology	10	10	0
Total	144	1640.5	1606

Mandatory Requirement 5

ID M.R.5

Context

Expand Phase I Project by adding the ability to sort by a user selected list.

Description

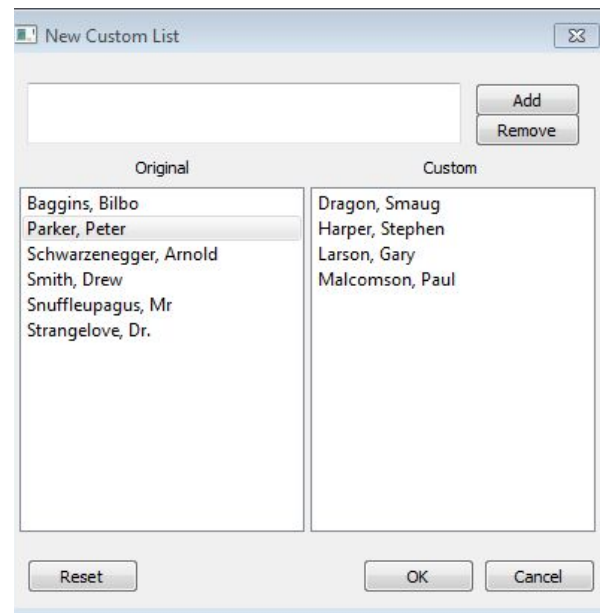
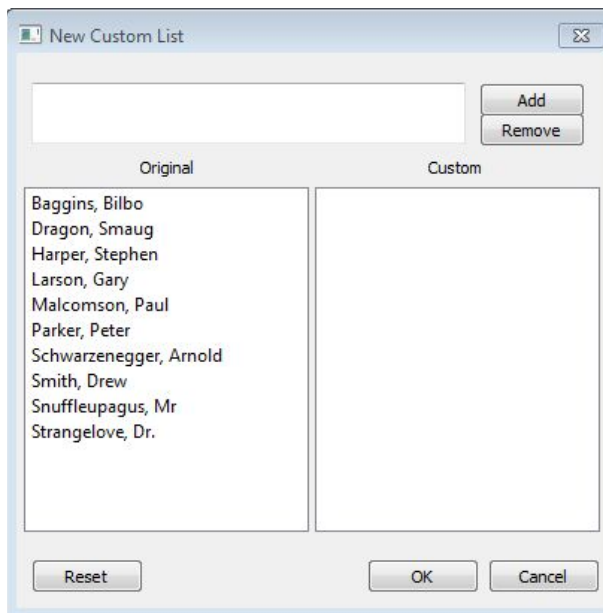
We implemented the ability to sort by a user selected list function. In Phase II, we created the file selectData.cpp and the interface for the new custom list. The user can click the “Custom List” button and then it will open a new window to show the list of member names. The user can remove or add the names from the list via double clicking the names or typing in a name in the format “LastName, FirstName” and click “Add” or “Remove”. After clicking “OK” to confirm, the main window will show only the selected member’s data. If the user wants to view the full member list again they, they can click the “Reset” button in the “Custom List” window. This user selected list function can be used for all the CSV file types: Teaching, Publications, Presentations and Funding.

Origin

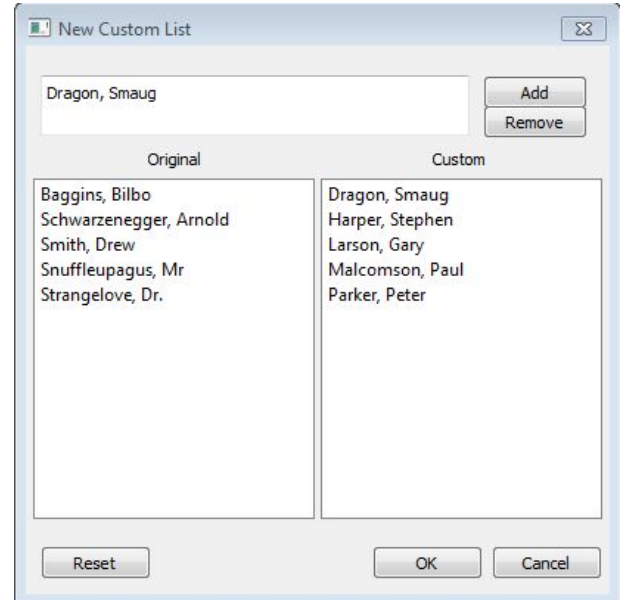
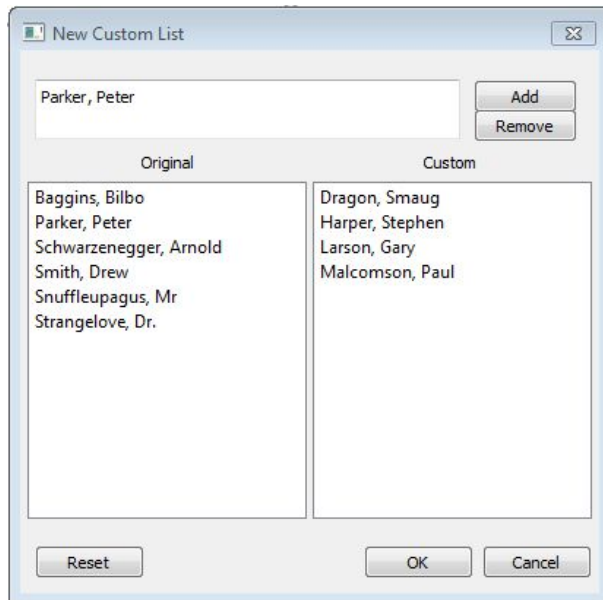
This was a mandatory requirement outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Example

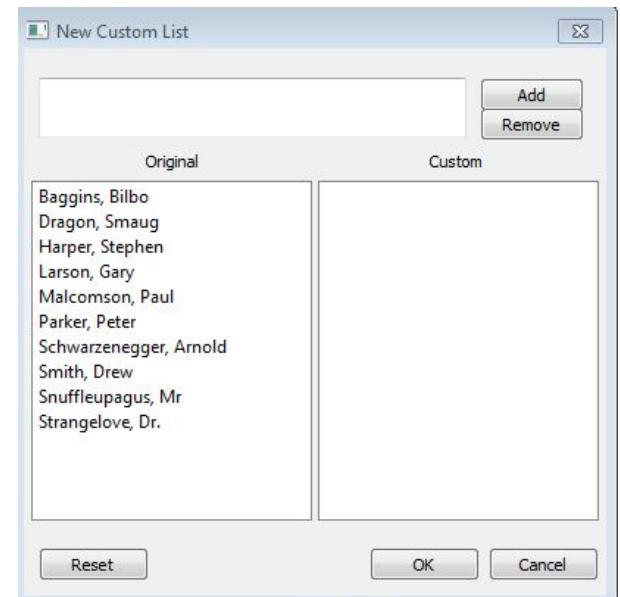
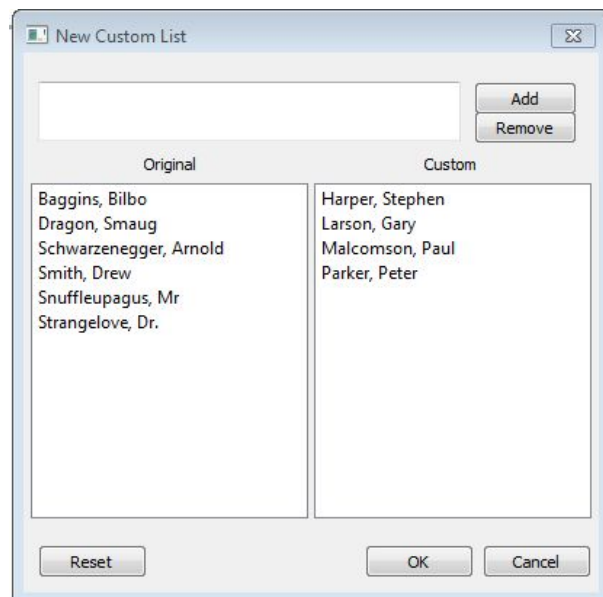
We will show how to select data by users. First, we click the “Custom List” button on the main window, and it will open a new window. From here we can add some names via double clicking.



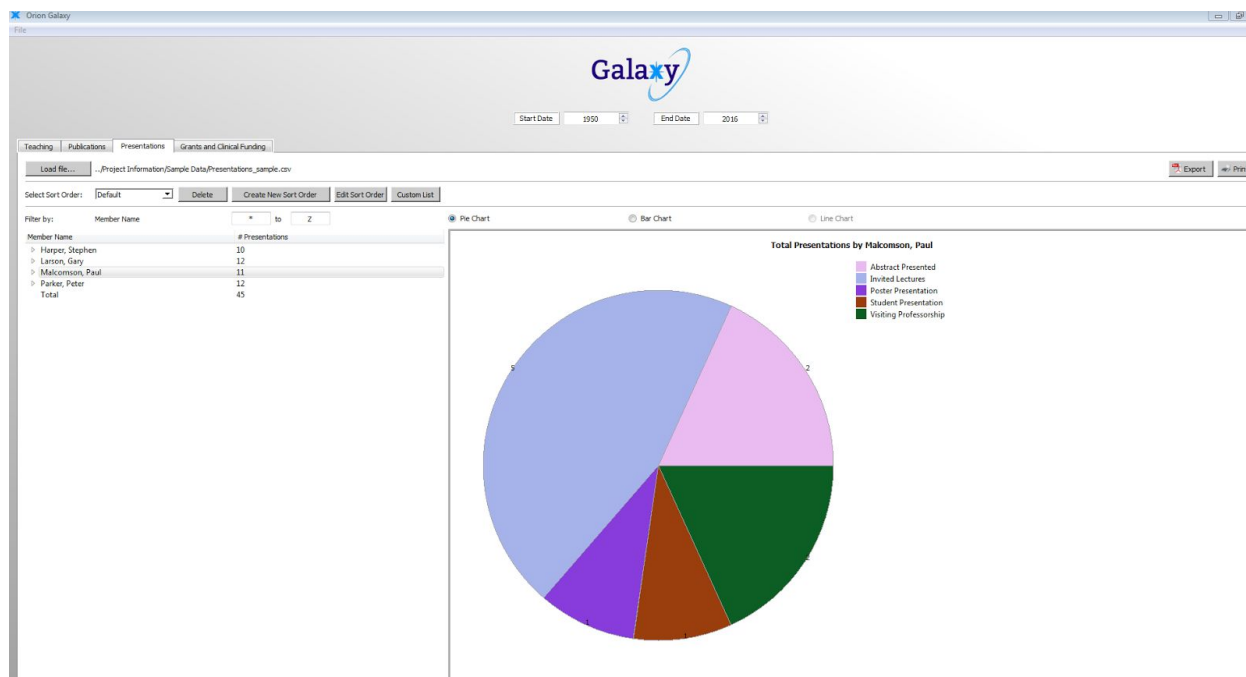
Alternatively, typing “Parker, Peter” and clicking “Add” will add the member to the custom list, and typing “Dragon, Smaug” and clicking “Remove” will remove the member from the custom list.



As an example, a list of the four names in the “Custom” window will be used. If the user would like to revert back to the original list, they can click “Reset”. Once the list is complete, click “OK”.



The main window will only show the selected data.



Mandatory Requirement 6

ID M.R.6

Context

Part of the Customer Specification

6. Fix all errors so that the finished product is error free.

Description

The program has been thoroughly tested and is not known to crash or produce errors.

Mandatory Requirement 7

ID M.R.7

Context

Part of the Customer Specification

7. Deliver a “User Proof” install file for final project containing
- a) Executable
 - b) Documentation
 - c) Supporting Files

Description

The executable installer file along with the installation guide, user guide, and video demo can be found in the Deliverable 5 folder (and have also been submitted separately).

Mandatory Requirement 8

ID M.R.8

Context

Part of the Customer Specification

8. Verify the code deck for operation on Windows 7
Alternatively Windows 10 could be used, if you wish to
use Win10 please specify this in your report

Description

The program has been tested successfully on windows 7 & 10.

Mandatory Requirement 9

ID M.R.9

Context

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.

Description

A feature in the Phase 1 system was that the user had the option to fix erroneous fields within the spreadsheet before use in the system. This included clicking 1 by 1, the different blocks where mandatory headers coincide. Beside the ability to enter data into these test fields, the only other 2 options were "save" and "close" where the user could save all the data they inputted but only if every data field was filled with text, and close to end the process. The customer specified that there should be a way to go through the fields without searching for them using the scrolling icons and clicking them.

We created two new methods within the ErrorEditDialog.cpp class called on_findNext_clicked() and on_findPrev_clicked() along with the implementation of 2 new ui buttons that corresponded to these methods. These methods looked at the current field you were in and determined where the "next" or "previous" error is located, and take the user there.

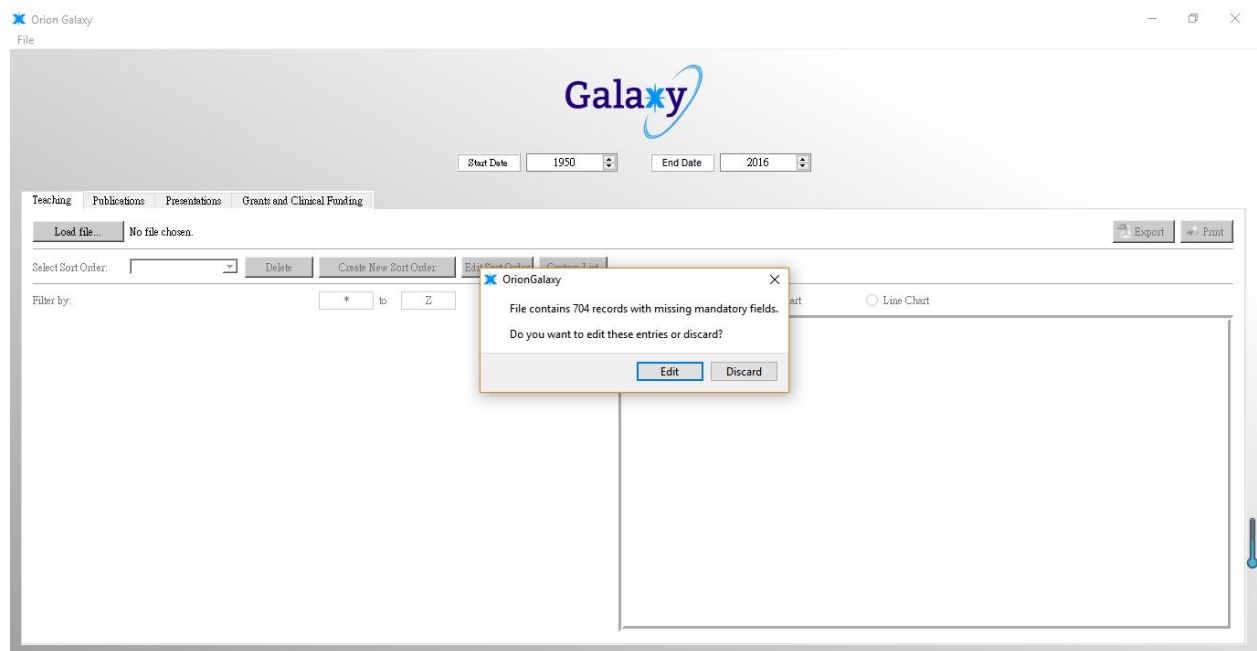
Along with this, a small text box is also displayed showing the user the amount of errors remaining after each fill. We calculated the number of errors by counting each block in the field and wherever there was an error we incremented the count. At first the methods would go through all columns and rows to do this which caused more processing power to take place and thus "lag" between switching fields. We then decided to create an array with the columns indexes where the mandatory requirements would be and used this as part of our search. Because of this there is now only a negligible delay to find the next error cell.

Origin

This was a mandatory requirement outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Example

First, load the file. Then it will have a window to ask user if he/she wants to edit the data



After pressing “edit, the user can edit the file. To navigate between missing fields and errors, a user can use the buttons “Find Next” and “Find Previous” which will take the user to either the next or previous error. Below it, it shows the amount of remaining errors each time the user enters data. After finishing edit, then press ok to save it.

Edit Erroneous Fields

	ID	Member Name	Department	Division	Start Date	End Date	Program	e of Cc
1	4697	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
2	4696	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
3	4695	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
4	4694	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
5	4693	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
6	4692	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
7	4691	Teefy, Patrick	Medicine	Cardiology			Continuing Me...	
8	6151	Jablonsky, Geor...	Medicine	Cardiology		1993/06/09	Continuing Me...	Present
9	6150	Jablonsky, Geor...	Medicine	Cardiology		1992/01/29	Continuing Me...	Present
10	6149	Jablonsky, Geor...	Medicine	Cardiology		1989/02/03	Continuing Me...	Present
11	6148	Jablonsky, Geor...	Medicine	Cardiology		1988/06/17	Continuing Me...	Present
12	6076	Jablonsky, Geor...	Medicine	Cardiology		1986/10/23	Continuing Me...	Present

Find Previous

Find Next

Remaining Errors:

1333

Save

Cancel

Stretch Goal 2 - Change / Modify Sort Order

ID S2

Context

Change / Modify Sort Order.

Description

We implemented the modify sort order function in the program. In the dashboard, we added an “Edit Sort Order” button beside the “Create New Sort Order” button, and a QT designer form class. When the user clicks the “Edit Sort Order” button, an edit sort window will appear, and the user can modify the current sort order in this window. After the user finishes modifying the current sort order, they can click the save button and the current sort order will be saved. After the user edits the current sort order, both the sorted list and the graph will be changed according to the sort list that the user edited. This required learning how to make use of the QT Designer to add widgets and bind functionality to the ‘slots’ of GUI elements, and also how to create a QT designer form class. We added 4 methods to the “MainWindow” class that allow user to modify the sort order, and also a QT designer form class which is named “Edit Sort”.

Origin

This was a stretch goal outlined by the customer (the Schulich School of Medicine and Dentistry) in their Phase II Goals document.

Example

We will show the edit sort order function by creating a new sort order and then modify it. First we create a sort order named “TestEdit” and the sort order is Member name, start date, program and division.

Create New Sort Order

New Custom Sort Order

Enter name of new custom sort:

TestEdit

Select order:

Member Name

Start Date

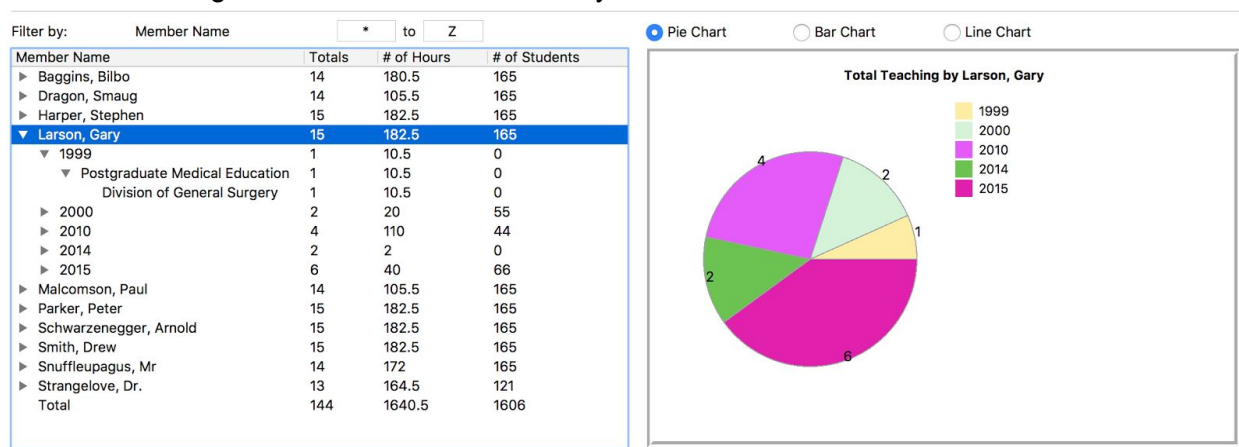
Program

Division

Cancel

Save

The list on the right hand side is determined by the “TestEdit” order.



Then we click the Edit Sort Order button, and the edit sort order window will appear.

Dialog

Edit Custom Sort Order

Program

Start Date

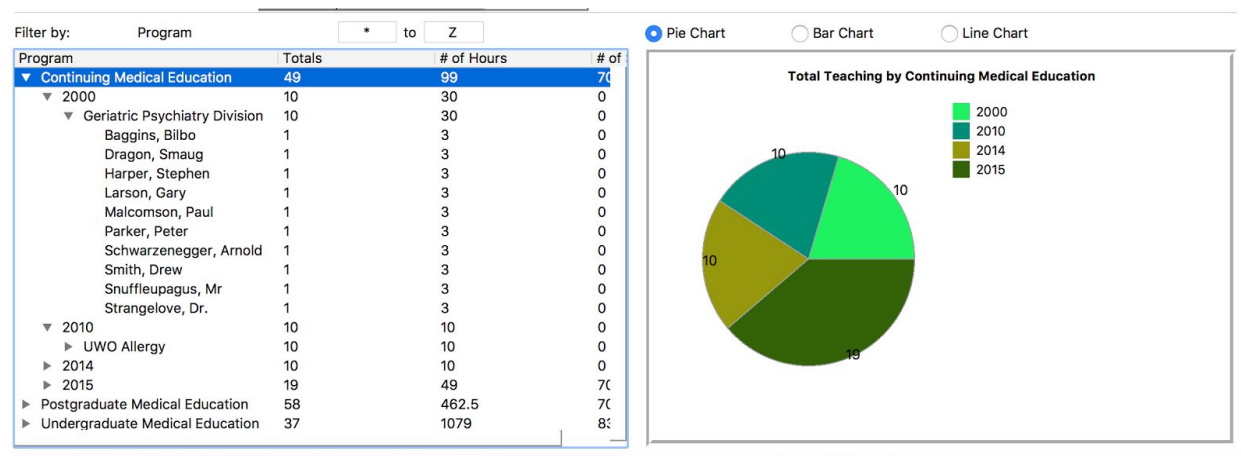
Division

Member Name

Cancel OK

Then we select the new order, which should be different from the original order. For example, we select program, start date, division and member name, and then click OK button. The order we edited is saved after we click the OK button.

When we go back to the dashboard, we can see the list and graph is changed to match the order we edited.



Stretch Goal 3 - Training Video

Team Orion has created a training video for any new user wishing to utilize the new features of the software. The video goes in depth of how to use the software, specifically the new features that were added. The video can be found within the installation package.

Test Cases for Requirements

Test cases for requirement ID M.R.1

The following tests were implemented to test the Phase I code base.

Test ID A:

Test 1 tests QSortListIO. This test will test saveList and readList functions in QSortListIO class, by first, creating a QStringList to store the order, then use saveTestSort function to save the order, then will implement readList function to get the order, if the order we get is same with the order we saved, then the test will pass.

Test ID B:

Test 2 Verifies that the RecordsManager constructor populates the correct headers from the .csv input file.

Test ID C:

Test 3 Verifies that the TeachingTreeModel constructor successfully creates a TeachingTreeModel object.

Test ID D:

Test 4 Verifies that the GrantFundingTreeModel constructor successfully creates a GrantFundingTreeModel object.

Test ID E:

Test 5: Verifies that the PresentationTreeModel constructor successfully creates a PresentationTreeModel object.

Test ID F:

Test 6: Verifies that the PublicationTreeModel constructor successfully creates a PublicationTreeModel object.


```

1  #include "Test.h"
2
3  //QSortListIO
4  void Test::test1()
5  {
6      QStringList testOrder;
7      testOrder << "Default";
8      QList<QStringList> allTestOrders;
9      allTestOrders << testOrder;
10     QSortListIO saveTestSort("testsortorder.dat");
11     saveTestSort.saveList(allTestOrders);
12     QList<QStringList> read = saveTestSort.readList();
13     QVERIFY(saveTestSort.readList() == allTestOrders);
14 }
15
16 //Test RecordsManager
17 void Test::test2(){
18     CSVReader reader("../Project Information/Sample Data/Grants_expanded.csv");
19     std::vector<std::string> header = reader.getHeaders();
20     RecordsManager* testRM = new RecordsManager(&header);
21     QVERIFY(testRM->getHeaders() == header);
22 }
23
24 //Test TeachingTreeModel
25 void Test::test3(){
26     CSVReader reader("../Project Information/Sample Data/Teaching_sample.csv");
27     std::vector<std::string> header = reader.getHeaders();
28     RecordsManager* testRM = new RecordsManager(&header);
29     TeachingTreeModel* teachTree = new TeachingTreeModel(testRM);
30     QVERIFY(teachTree!=NULL);
31 }
32
33 //Test GrantFundingTreeModel
34 void Test::test4(){
35     CSVReader reader("../Project Information/Sample Data/Grants_expanded.csv");
36     std::vector<std::string> header = reader.getHeaders();
37     RecordsManager* testRM = new RecordsManager(&header);
38     GrantFundingTreeModel* grandTree = new GrantFundingTreeModel(testRM);
39     QVERIFY(grandTree!=NULL);
40 }
41
42 //Test PresentationTreeModel
43 void Test::test5(){
44     CSVReader reader("../Project Information/Sample Data/Presentations_expanded.csv");
45     std::vector<std::string> header = reader.getHeaders();
46     RecordsManager* testRM = new RecordsManager(&header);
47     PresentationTreeModel* presentationTree = new PresentationTreeModel(testRM);
48     QVERIFY(presentationTree!=NULL);
49 }
50
51 //Test PublicationTreeModel
52 void Test::test6(){
53     CSVReader reader("../Project Information/Sample Data/Publications_expanded.csv");
54     std::vector<std::string> header = reader.getHeaders();
55     RecordsManager* testRM = new RecordsManager(&header);
56     PublicationTreeModel* publicationTree = new PublicationTreeModel(testRM);
57     QVERIFY(publicationTree!=NULL);
58 }
59
60

```

Test ID G:

Additional test cases 7 - 14 were constructed to test the CSV reader.

```

80
81 //Tests for CSV Reader
82 void Test::test7(){
83     CSVReader reader("../Project Information/Sample Data/Grants_expanded.csv");
84     vector<string> header = reader.getHeaders();
85     QVERIFY2(header.size() !=0, "Test1 Failed");
86 }
87
88 void Test::test8(){
89     CSVReader reader("../Project Information/Sample Data/GrantsClinicalFunding_sample.csv");
90     vector<string> header = reader.getHeaders();
91     QVERIFY2(header.size() !=0, "Test2 Failed");
92 }
93
94 void Test::test9(){
95     CSVReader reader("../Project Information/Sample Data/Presentations_expanded.csv");
96     vector<string> header = reader.getHeaders();
97     QVERIFY2(header.size() !=0, "Test3 Failed");
98 }
99
100 void Test::test10(){
101     CSVReader reader("../Project Information/Sample Data/Presentations_sample.csv");
102     vector<string> header = reader.getHeaders();
103     QVERIFY2(header.size() !=0, "Test4 Failed");
104 }
105
106 void Test::test11(){
107     CSVReader reader("../Project Information/Sample Data/Program_Teaching_expanded.csv");
108     vector<string> header = reader.getHeaders();
109     QVERIFY2(header.size() !=0, "Test5 Failed");
110 }
111
112 void Test::test12(){
113     CSVReader reader("NO FILE");
114     vector<string> header = reader.getHeaders();
115     QVERIFY2(header.size() ==0, "Test6 Failed");
116 }
117
118 void Test::test13(){
119     CSVReader reader("../Project Information/Sample Data/Presentations_sample.csv");
120     vector< vector<string> > all_data = reader.getData();
121     QVERIFY2(all_data.size() !=0, "Test7 Failed");
122 }
123
124 void Test::test14(){
125     CSVReader reader("../Project Information/Sample Data/Program_Teaching_expanded.csv");
126     vector< vector<string> > all_data = reader.getData();
127     QVERIFY2(all_data.size() !=0, "Test8 Failed");
128 }
129

```

All tests were verified and passed without issue.

```

Starting D:\Documents\Desktop\School\TeamOrion-master\AcuitySTAR\build-TeamPeach-Desktop_Qt_5_7_0_MinGW_32bit-Debug\debug\TeamPeach.exe...
***** Start testing of Test *****
Config: Using QTest library 5.7.0, Qt 5.7.0 (i386-little_endian-ilp32 shared (dynamic) debug build; by GCC 5.3.0)
PASS : Test::initTestCase()
PASS : Test::test1()
PASS : Test::test2()
PASS : Test::test3()
PASS : Test::test4()
PASS : Test::test5()
PASS : Test::test6()
PASS : Test::test7()
PASS : Test::test8()
PASS : Test::test9()
PASS : Test::test10()
PASS : Test::test11()
Couldn't open file "NO FILE".
PASS : Test::test12()
PASS : Test::test13()
PASS : Test::test14()
PASS : Test::cleanupTestCase()
Totals: 16 passed, 0 failed, 0 skipped, 0 blacklisted, 2479ms
***** Finished testing of Test *****

```

Test cases for requirement ID M.R.2

For deliverable 3, we wrote 5 tests cases for the line chart requirement, and ran them again here to make sure that the addition of extra features did not interfere with their proper functioning.

Test ID H:

Test_on_teach_line_button_toggled is to test if the line chart works in teaching.

Test ID I:

Test_on_pres_line_button_toggled is to test if the line chart works in presentations.

Test ID J:

Test_on_pub_line_button_toggled is to test if the line chart works in publications.

Test ID K:

Test_on_fund_line_button_toggled is to test if the line chart works in funding.

Test ID L:

test_setupLineChart is to test if line chart can be set up..

```
//test on_teach_line_button_toggled function
void Test::test_on_teach_line_button_toggled(){
    w.on_teach_line_button_toggled();
    QCOMPARE(w.ui->teach_graph_stackedWidget->currentIndex(),2);
}

//test on_pres_line_button_toggled function
void Test::test_on_pres_line_button_toggled(){
    w.on_pres_line_button_toggled();
    QCOMPARE(w.ui->teach_graph_stackedWidget->currentIndex(),2);
}

//test on_pub_line_button_toggled function
void Test::test_on_pub_line_button_toggled(){
    w.on_pub_line_button_toggled();
    QCOMPARE(w.ui->teach_graph_stackedWidget->currentIndex(),2);
}

//test on_fund_line_button_toggled function
void Test::test_on_fund_line_button_toggled(){
    w.on_fund_line_button_toggled();
    QCOMPARE(w.ui->teach_graph_stackedWidget->currentIndex(),2);
}

//test setupLineChart function
void Test::test_setupLineChart() {
    int size = 5;
    std::vector<std::pair<std::string, double>> chartList;
    for (int i = 0; i < size; i++) {
        chartList.emplace_back("test", static_cast<double>(0.0));
    }
    w.setupLineChart(w.ui->teachLineChart,chartList);
    QCOMPARE(w.ui->teachLineChart->plottableCount(),(int) chartList.size());
}
```

To add the fourth chart type, we added a final method, `setupScatterPlot`, which was also tested similarly.

Test ID M:

`test_setupScatterPlot` is to test if scatter plot can be set up.

```
//test setupScatterPlot function
void Test::test_setupScatterPlot() {
    std::vector<std::pair<std::string, double>> chartList;
    for (int i = 0; i < 5; i++) chartList.emplace_back("10",100000);
    w.ui->fundHistogramChart->setEnabled(true);
    w.setupScatterPlot(w.ui->fundHistogramChart,chartList);
    QCOMPARE(w.ui->fundHistogramChart->plottableCount(),1);
}
```

All of the tests generated for the new charts types were successful:

```
PASS : Test::test_on_teach_line_button_toggled()
PASS : Test::test_on_pres_line_button_toggled()
PASS : Test::test_on_pub_line_button_toggled()
PASS : Test::test_on_fund_line_button_toggled()
PASS : Test::test_setupLineChart()
PASS : Test::test_setupScatterPlot()
```

Test cases for requirement ID M.R.3 - Add save session

Test ID N:

`NoSaveTest` is to test if there is a saved data or not.

Test ID O:

`SaveTestTeach` is to test if the user can save data for teaching.

Test ID P:

`SaveTestPub` is to test if the user can save data for publications.

Test ID Q:

`SaveTestPres` is to test if the user can save data for presentations.

Test ID R:

`SaveTestFund` is to test if the user can save data for funding.

```

void Test::NoSaveTest(){
    MainWindow w;
    w.show();
    w.close();
    QFileIO teachFile("teachfile.dat");
    QFileIO pubFile("pubfile.dat");
    QFileIO presFile("presfile.dat");
    QFileIO fundFile("fundfile.dat");
    QString teachPath = teachFile.readPath();
    QString pubPath = pubFile.readPath();
    QString presPath = presFile.readPath();
    QString fundPath = fundFile.readPath();
    QVERIFY2(teachPath == "" && pubPath == "" && presPath == "" && fundPath == "", "No save test failed");
}

void Test::SaveTestTeach(){
    MainWindow w;
    w.show();
    w.load_teach("../Project Information/Sample Data/Program Teaching_expanded.csv");
    w.close();
    QFileIO teachFile("teachfile.dat");
    QString readTeachPath = teachFile.readPath();
    QVERIFY2(readTeachPath != "", "Save Test for Teaching failed");
}

void Test::SaveTestPub(){
    MainWindow w;
    w.show();
    w.load_pub("../Project Information/Sample Data/Publications_sample.csv");
    w.close();
    QFileIO pubFile("pubfile.dat");
    QString readPubPath = pubFile.readPath();
    QVERIFY2(readPubPath != "", "Save Test for Publications failed");
}

void Test::SaveTestPres(){
    MainWindow w;
    w.show();
    w.load_pres("../Project Information/Sample Data/Presentations_sample.csv");
    w.close();
    QFileIO presFile("presfile.dat");
    QString readPresPath = presFile.readPath();
    QVERIFY2(readPresPath != "", "Save Test for Presentations failed");
}

void Test::SaveTestFund(){
    MainWindow w;
    w.show();
    w.load_fund("../Project Information/Sample Data/GrantsClinicalFunding_sample.csv");
    w.close();
    QFileIO fundFile("fundfile.dat");
    QString readFundPath = fundFile.readPath();
    QVERIFY2(readFundPath != "", "Save Test for Funding failed");
}

```


Test result:

```
PASS : Test::SaveTestTeach()
PASS : Test::SaveTestPub()
PASS : Test::SaveTestPres()
PASS : Test::SaveTestFund()
PASS : Test::testSortByDivision()
PASS : Test::test on teach line button toggled()
```

Test cases for requirement ID M.R.4 - sort by member's division

Test ID S:

TestSortByDivision is to test if the user can sort data by division.

```
void Test::testSortByDivision()
{
    bool testPasses = false;
    //QString path = "../Project Information/Sample Data/Teaching_sample.csv";
    //w.load_teach(path, false);    //load teaching file

    QStringList newSortOrder = (QStringList() << "Division" << "Program"); //create new sort order (simulates w.on_teach_new_sort_clicked())

    w.allTeachOrders << newSortOrder;
    w.ui->teach_sort->addItem(newSortOrder.at(0)); //add new sort order to mainwindow attributes

    for (int i=0; i < w.allTeachOrders.size(); i++) //check if sort orders contain a tier for "Division"
    {
        QStringList qsl = w.allTeachOrders.at(i);
        if (qsl.contains("Division"))
        {
            testPasses = true;
            //maybe could also be done by evaluating the tree model
        }
    }

    QVERIFY(testPasses);
}
```

Test result:

```
PASS : Test::SaveTestPub()
PASS : Test::SaveTestPres()
PASS : Test::SaveTestFund()
PASS : Test::testSortByDivision()
PASS : Test::test on teach line button toggled()
PASS : Test::test on pres line button toggled()
PASS : Test::test on pub line button toggled()
PASS : Test::test on fund line button toggled()
```

Test cases for requirement ID M.R.5 - Create Custom List

Test ID T:

testCustomList() is a UI test element for creating a custom list

```
253 void Test::testCustomList() {
254     std::vector<std::string> testFields;
255     testFields.push_back("test");
256     std::string testString = "test";
257     selectData* sortdialog = new selectData();
258     sortdialog->setFields(testFields);
259     sortdialog->charInField=testString;
260     sortdialog->selectData::on_addButton_clicked();
261     sortdialog->selectData::on_buttonBox_accepted();
262     QCOMPARE(sortdialog->getCustomSortFields(), testFields);
263 }
264
```

Test result:

```
PASS : Test::testCustomList()
```

Test cases for requirement ID M.R.6 - Correct all errors

Test ID U:

The new csv files had new date formats which was causing records not to be loaded. This tests that all records are loaded properly.

```
void Test::testFixDateFormatting()
{
    CSVReader reader("../Project Information/Sample Data/Grants_expanded.csv");
    int numRecords = reader.getData().size();
    QVERIFY(numRecords == 4241);
}
```

Test result:

```
PASS : Test::testFixDateFormatting()
```

Test cases S.R.2 - Edit sort order

Test ID V:

test_editsort_setFields() is a UI test element for the new feature.

Test ID W:

test_editsort_getSortFields() is a UI test element for the new feature.

```

void Test::test_editsort_setFields(){
    std::vector<std::string> testString;
    testString.push_back("a");
    editui.setFields(testString);
    QCOMPARE(editui.fieldBoxes.at(0)->currentText(),QString::fromStdString("a"));
}

void Test::test_editsort_getSortFields(){
    std::vector<std::string> testString;
    editui.setFields(testString);
    QStringList testList;
    QCOMPARE(editui.getSortFields(),testList);
}

```

Test ID X:

test click the cancel button on edit sort order window.

```

void Test::test_on_buttonBox_rejected(){
    editui.on_buttonBox_rejected();
    QCOMPARE(editui.isActiveWindow(),false);
}

```

Test ID Y:

test click the button for edit sort order window.

```

void Test::test_on_edit_button_clicked(){
    QTest::keyClicks( w.ui->teach_edit_sort, "Edit Sort Order");
    QCOMPARE( w.ui->teach_edit_sort->text(), QString("Edit Sort Order"));
}

```

Test result:

```

PASS    : Test::test_editsort_setFields()
PASS    : Test::test_editsort_getSortFields()
PASS    : Test::test_on_buttonBox_rejected()
PASS    : Test::test_on_edit_button_clicked()

```


Testing Matrix

The following matrix represents all of the requirements implemented to date organized by ID along with the test cases involved in the testing of each.

ID	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
MR. 1	X	X	X	X	X	X	X																		
MR. 2								X	X	X	X	X	X												
MR. 3														X	X	X	X	X							
MR. 4																			X						
MR. 5																				X					
MR. 6																					X				
SR.2																						X	X	X	X

A = test1, QSortListIO

B = test2, RecordsManager

C = test3, TeachingTreeModel

D = test4, GrantFundingTreeModel

E = test5, PresentationTreeModel

F = test6, PublicationTreeModel

G = test7-14 all test the CSV reader included in Phase I, and have been combined here because of their similarity and to save space on the matrix

H = test_on_teach_line_button_toggled

I = test_on_pres_line_button_toggled

J = test_on_pub_line_button_toggled

K = test_on_fund_line_button_toggled

L = test_setupLineChart

M = test_setupScatterPlot

N = NoSaveTest

O = SaveTestTeach

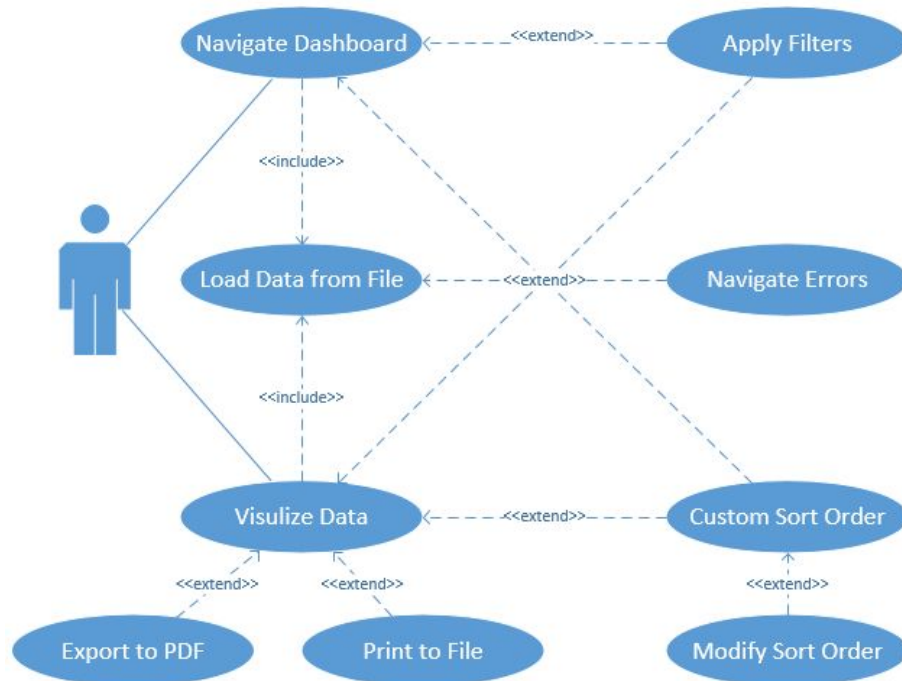
P = SaveTestPub

Q = SaveTestPres

R = SaveTestFund
S = TestSortByDivision
T = testCustomList
U = testFixDateFormatting
V = test_editsort_setFields
W = test_editsort_getSortFields
X = test_on_buttonBox_rejected
Y = test_on_edit_button_clicked

System Design

Use Case Diagram v3



Our Use-Case Diagram represents all the different ways that a user might interact with the Peach Galaxy system. The end user of the software will navigate the dashboard and use it to analyse and visualize the data contained in the CSV files. They will be able to load and modify these files, and once loaded, plot them and sort them according to their specific requirements. As Phase II is an extension of Phase I, these features and the use case diagram presented here is very similar to the use case diagram of Phase I. The addition of new graph types did not change the interaction where the user is able to visualise data, and the ability to sort the data is similar to before. There are, however, a few notable exceptions. The extension of the use case 'Load Data from File' has been renamed from 'Process errors' to 'Navigate errors', which satisfies the customer's mandatory requirement number 9. Phase I had labelled this as 'Process errors' but we found that that was more indicative of how things work on the software side and not how the user would interact with the system. Additionally, in this phase we have extended the 'Custom Sort Order' use case with the option to 'Modify Sort Order' to satisfy Stretch Goal 2. The other requirements implemented so far in Phase II are not reflected on the Use-Case diagram because they do not affect the user's interaction with the system, but rather modify the backend.

Class Diagram v3

[imgur link](#)



For this deliverable, we added the ability to view multiple users data at once. This required the addition of four new methods inside the 'MainWindow' class for each of the four file types. These were extensions of the currently present methods that convert data from the index which stores all the data in the current view into chart lists that can then be plotted on each of the four graph types. Additionally, we added the ability to visualize funding data using a scatter plot so we added the method 'setupScatterPlot' inside the 'MainWindow' class. This method was easier to implement after having learned in the previous deliverable how 'qcustomplot' works.

The new csv file formats used and erratic date format (eg. 16-Jul, Feb-14 instead of 2016-03-21) which was causing the program to be unable to load the files. In order to accommodate the new csv file format, a new method fixDateFormatting was implemented in CSVreader.cpp in order to convert the new dates into a usable format. This (along with some logic improvements to the recursive analyze method in RecordsManager.cpp) allowed the program to load files in the new format.

For M.R.3 (Ability to save session state) a class called QFileIO was created to accept a QString of the imported file's path and serialize it to a .dat file which is called by Mainwindow when the program is reopened.

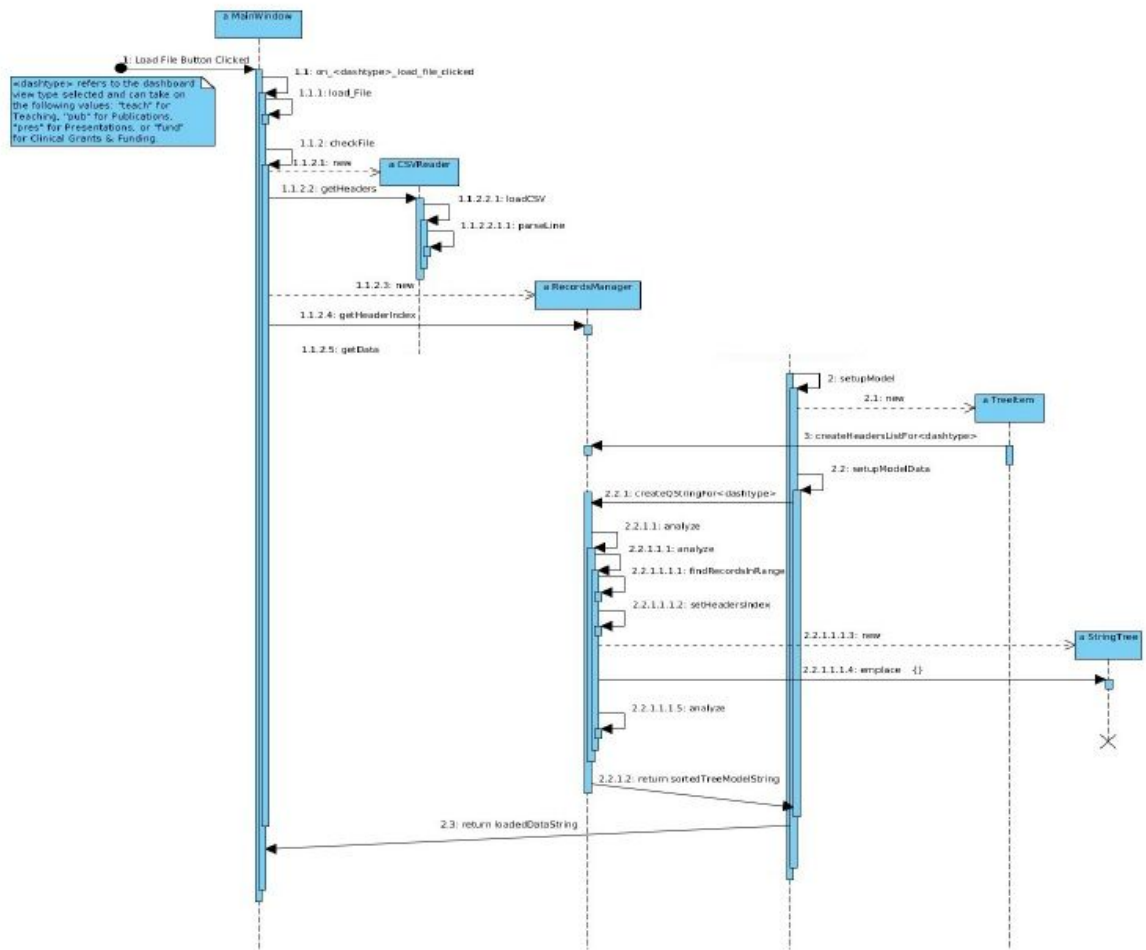
For M.R.5 (Ability to sort by user selected list) a class called selectData was created to accept a vector of strings of member names for the user to choose to add to a Custom List and display information of only the members on that list.

For M.R.9 (Error Edit Navigation) 3 new methods were added to ErrorEditDialog.cpp countRemainingErrors() to count the cells that still need to be populated on _FindNext_clicked() & on _FindPrev_clicked in order to navigate forwards & backwards through the errors. This design allowed for a relatively direct and simplistic way to implement this requirement.

For S.R.2(Edit sort order), the EditSort class is added. The class defines the edit sort order GUI component. It allows user edit current order.

Sequence Diagram v2

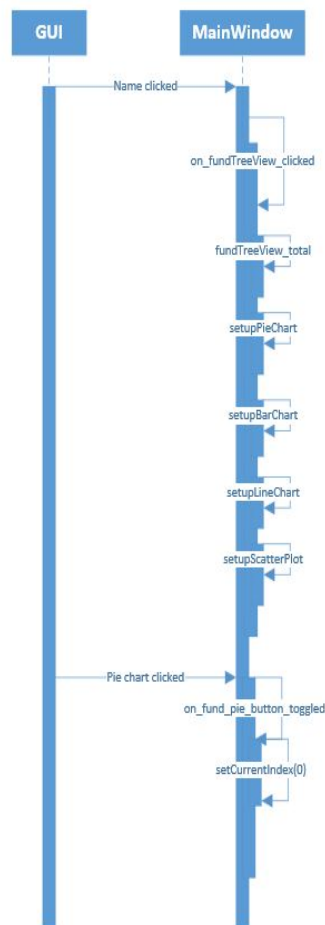
The sequence diagram above is closely related to the old one from the phase 1 system. This is because the requirements implemented are so closely related to the phase 1 objects that there would be no need to show another path for the same process (i.e. line & scatter plot graph, added the “Division” category for sort and save session). Nonetheless, with the future requirements to be implemented in the system we will see a change within certain diagrams, such as the sequence diagram. Eg, within the CSVReader there is a method within to extract the proper year from the date. This method would be seen between loadCSV and parseLine coming from the CSVReader method onto itself just like the “LoadCSV” method.



Context: Sequence of actions taken for the user to be displayed the Error Dialog Box when a file has been inputted. When the user in this situation says “yes” to fix errors within the fields of the file and also clicking on “next” each time an error has been fixed.

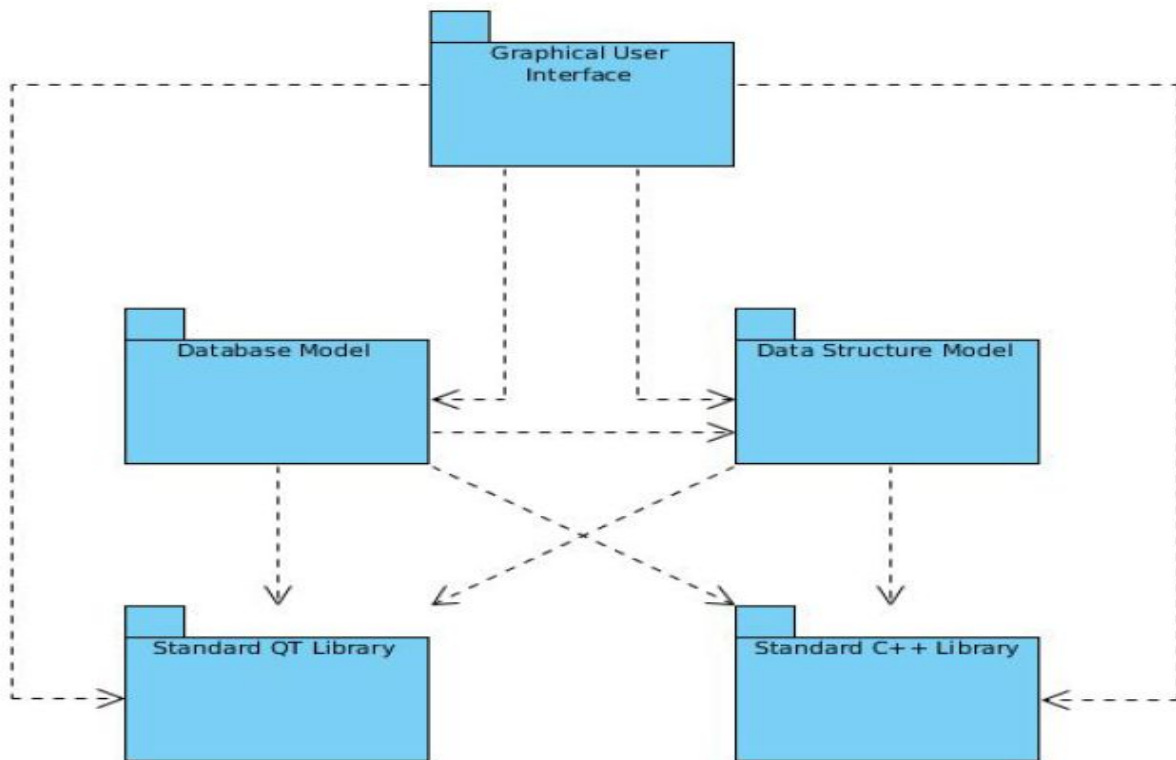


Sequence Diagram #3 - Visualization with Graphs



When a user wants to visualize certain data types using one of the graph types included with the program, they must simply click on the type of graph that they wish to use. However, it is interesting to note that the graphs are not created as a user clicks on them. When a name on the list in the left hand side pane is clicked, a sequence of events on the backend is triggered. First the program generates lists containing the relevant information. Then these lists are passed to the empty frames for each chart type. In this program we make heavy use of the `QCustomPlot` class, which contains lots of methods for making charts, adding new data and of course making them look nice. As a result of these steps, even when one of the graph types is not currently selected, the information is already loaded and ready to go, enabling faster switching. Then, as soon as a new graph type is toggled, an index is updated to switch the current view.

Package Diagram v2

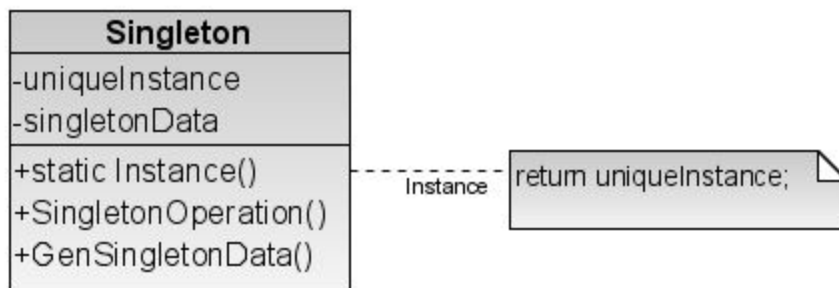


This is the package diagram for Peachy Galaxy system in Phase II, which is basically the same one as was used in Phase I. This shows all the packages and their dependencies. In the diagram, the dotted arrows from the source package to the target package represents a dependency. There are five packages in this diagram, Graphical User Interface, Database Model, Data Structure Model, Standard QT Library and Standard C++ Library. The Graphical User Interface package contains the classes such as MainWindow, PieChartWidget, CustomSort and SelectData. These classes use both the standard QT and C++ libraries to run the graphical components. In addition, the Graphical User Interface package relies on Database Model and Data Structure Model packages in order to create new database and data structure models. Meanwhile, those two packages also need to use the standard QT and C++ libraries. There was only more code added to specific classes to handle new requirements. Because these requirements required no new libraries then there was no reason to change up the package diagram.

Design Patterns

Singleton

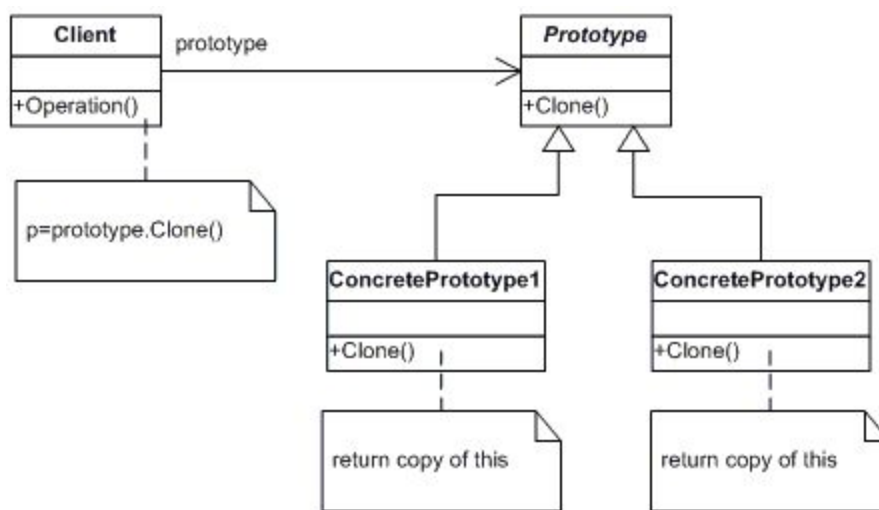
The singleton design pattern suggests that a class only be instantiated once and only once and provide a global point of access. This class would ensure that no other instance can be created. In a situation it can be important for a class to have only one instance as others for a particular type of program can cause errors or bugs. The best solution for that class to manage that only one instance is created is having that same class be responsible for keeping track of its sole instance.



The software of “Peachy Galaxy” uses this design pattern although with more flexibility in terms of modifying the instance should it become necessary. The implementation of `MainWindow()`, which controls the runtime behaviour of the graphical user interface, is the class in which the singleton pattern can be utilized. With this in place only one instance of the class is created and accessible to clients from a well-known access point. How it works is to hide the operation that creates the instance behind a class operation (that is, either a static member function or a class method) that guarantees only one instance is created. The class in C++ is defined with a static member function `instance` of the singleton class. Singleton also defines a static member variable `unique Instance` that contains a pointer to its unique instance.

Prototype

This design pattern was also utilized along with the singleton in the implementation of this project. The basic goal of the prototype pattern is to specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype. Suppose the system has many objects which, although differ slightly from each other, exhibit almost identical behaviour. The idea is to follow the last team's design in that, because the object composition is a flexible alternative to subclass, we make the framework take advantage of this by cloning or copying an instance of the desired class. We call this instance a prototype and depict its structure below.



By using this design pattern we can:

- 1) *Adding and removing products at run-time*: prototypes allow the incorporation of a new concrete product class into a system simply by registering a prototypical instance with the client. This is slightly more flexible than other creational patterns because a client can install and remove prototypes at run-time.
- 2) *Specifying new objects by varying values*: highly dynamic systems permit new behavior through object composition—by specifying values for an object's variables, for example—and not by defining new classes. One effectively defines new kinds of objects by instantiating existing classes and registering the instances as prototypes of client objects. A client can exhibit new behavior by delegating responsibility to the prototype. This kind of design lets users define new "classes" without programming. In fact, cloning a prototype is similar to instantiating a class. The Prototype pattern can greatly reduce the number of classes a system needs.
- 3) *Specifying new objects by varying structure*: many applications build objects from parts and subparts. Our graphical user interface, for example, is built from widgets, dialog boxes, radio buttons, etc. For convenience, such applications often let one instantiate complex, user-defined structures, say, to use a specific widget again and again. The Prototype pattern supports this as

well. We simply add this widget as a prototype to the palette of available graphical user interface elements.

- 4) *Reduced subclassing*: other alternatives to the Prototype pattern, such as the Factory Method, often produce a hierarchy of Creator classes that parallels the product class hierarchy. The Prototype pattern allows one to clone a prototype instead of asking a factory method to make a new object. Hence one does not need a Creator class hierarchy at all. This benefit applies primarily to languages like C++ that don't treat classes as first-class objects.

The main liability of the Prototype pattern is that each subclass of Prototype must implement the Clone operation, which may be difficult. For example, adding Clone is difficult when the classes under consideration already exist. Implementing Clone can be difficult when their internal include objects don't support copying or have circular references.

In the case of TreeModel, we use what is referred to as a prototype manager. When the number of prototypes in a system isn't fixed (that is, they can be created and destroyed dynamically), we keep a registry of available prototypes. Clients will not manage prototypes themselves but will store and retrieve them from the registry. A client will ask the registry for a prototype before cloning it. Unfortunately our prototype classes do not define operations for (re)setting key pieces of state. If not, then you may have to introduce an initialization operation that takes initialization parameters as arguments and sets the clone's internal state accordingly. An example of this is the `setupModel()` operation in `Treeltem`.

The design of the GUI in the `MainWindow` class also makes heavy use of a variant of the Prototype pattern in the design of its various private and public methods. This behaviour has carried over from the Peach Galaxy phase of the program into our version of the system. Consider the four main tabs of the program, namely Teaching, Publications, Presentations and the Grants/Funding page. Each of these tabs has a very similar behaviour: they all allow the user to click on names and view the data relevant to those names. The only difference is what type of CSV was loaded, which determines exactly what fields are available for analysis. Thus, the code used in each of these tabs is very similar: they all include a variation of `on_treeView_clicked`, which triggers the setup of all the different graph types via `setupPieChart`, `setupBarChart`, `setupLineChart` and `setupScatterPlot`, while also calling on a variation of the method `treeViewTotal`. Additionally, to load files from the program, each tab includes its own slight variation of the loading method: `load_fund`, `load_pub`, `load_teach` and `load_pres`.

Code and Design Inspection

Instructions:

- The purpose of this document is to assist in the inspection of object-oriented design.
- Under each question is a choice of answers; please choose one (either replace the box with a checkmark or highlight it)
 - ☐ yes
 - ☐ no
 - ☐ partly, could be improved
- Two types of comments are required under each question: (i) your analysis, and (ii) your finding (in the form of a comment). The analysis would typically show how you arrived at the finding.
- Add new lines as necessary for your analysis or findings.

Scope and process:

- Choose a subset of the system's features.
- Create some scenarios in which these features will be used.
- Keep the inspection process down to, say, two hours. This gives you some preliminary experience with inspection.
- Log improvement suggestions.
- Make improvements as appropriate.

+++++

Classes/Methods Inspected: MainWindow, setupLineChart, setupScatterPlot and the various slots that are required to activate these methods

Performed by: Christopher Brown

Structural correspondence between Design and Code:

Are all the classes and interrelationships programmed in the application explicitly represented in the class diagram of the system?

☒ Yes ☐ No ☐ Partly (Can be improved)

Comment on your analysis: A diagram was generated using another diagram software to aid with this. By inspection, all classes in the codebase are included in the diagram

Comment on your findings: All the Classes were found to be within the class diagram along with the corresponding relationships.

Functionality:

Do all the programmed classes perform their intended operations as per the requirements?

☒ Yes o No o Partly (Can be improved)

Comment on your analysis: Yes. See the testing section of the document for further details. on_treeViewClicked calls upon all the functions that are used to create the different types of graphs added to the program, namely setupLineChart and setupScatterPlot.

Comment on your findings: The graph types can get hard to decipher when there is a large amount of data being displayed at once.

Cohesion:

Do the methods encapsulated in each programmed class, together perform a single, well defined, task of the class? (High-Cohesion (good): the functionalities embedded in a class, accessed through its methods, have much in common, e.g., access common data)

☒ Yes o No o Partly (Can be increased)

Comment on your analysis: Besides the slots, which by necessity included a lot of subtasks, the methods included are very standalone and operate only within the class itself.

Comment on your findings: setupLineChart and setupScatter plot only modify information within the MainWindow class.

Coupling:

Do the programmed classes have excessive inter-dependency? (High Coupling (bad): In this case a class shares a common variable with another class, or relies on, or controls the execution of, another class.)

☐ Yes ☒ No o Partly (Can be reduced)

Comment on your analysis: The methods included for this requirement are all private so they do not affect the coupling of the program.

Comment on your findings: setupLineChart and setupScatter plot are not accessible outside the class.

Separation of concerns:

Is the scoped problem decomposed into separate concerns where each concern is encapsulated in a construct such as a class with well-defined interface and cohesive functions with minimal of connections with other concerns?

☒ Yes o No o Partly (Can be improved)

Comment on your analysis: During the design phase of the project, we made sure to try and break the code up as much as possible.

Comment on your findings: The methods that setup graph types only do that. Others are used to generate the data for them.

Do the classes contain proper access specifications (e.g.: public and private methods)?

☒ Yes o No o Partly (Can be improved)

Comment on your analysis: During the testing phase, we looked over the program to make sure this requirement was fulfilled.

Comment on your findings: In any class declaration, the methods are properly divided up between the private and public subsections as required.

Reusability:

Are the programmed classes reusable in other applications or situations?

☐ Yes, most of the classes o No, none of the classes ☒ Partly, some of the classes
o Don't know

Comment on your analysis: Even as we developed the program we felt that some of the code could be refactored so as to be more reusable, but a lot of this was part of the phase 1 system which would have taken a long time to rewrite.

Comment on your findings: From the phase 1, there are four tabs in the program, each of which could have made more use of OO principles. There is a lot of copy and pasting, and so as a result, the code for graphs uses a lot of copy and pasting and less reusability in order to integrate with the old code.

Simplicity:

Are the functionalities carried out by the classes easily identifiable and understandable?

☐ Yes o No ☒ Partly (Can be improved)

Comment on your analysis: The names of the functions and classes were chosen with care.

Comment on your findings: Names like setupLineChart and setupScatterPlot give a clear indication of what they do.

Do the complicated portions of the code have comments for ease of understanding?

☐ Yes ☒ No ☐ Partly (Can be improved)

Comment on your analysis: Understanding of the code implemented for this feature is not so difficult, it was the initial implementation that was harder to figure out.

Comment on your findings: Like in phase 1, there is a lack of comments in the software program. This is something that would need to be improved in future phases of the project.

Maintainability:

Does the application provide scope for easy enhancement or updates? (e.g., enhancement in the code is not anticipated to require too many changes in the original code)

☐ Yes ☒ No ☐ Partly (Can be improved)
☐ Don't know

Comment on your analysis: The MainWindow class in which this code was written interacts with a lot of classes, so it's hard to change without changing other classes.

Comment on your findings: At the same time, it would be relatively straightforward for future maintainers to add more graph types by following the example of the code already present.

Efficiency:

Does the design introduce inefficiency in code (e.g., causes too many nested loops or delays in concurrent processing)?

☒ Yes ☐ No ☐ Partly (Can be improved)
☐ Don't know

Comment on your analysis: The code required for this section does not require excessive computations, but is limited by the speed at which the program builds the data index out of the input CSV files.

Comment on your findings: Within methods like setupLineChart, there are for loops to get all the correct data, but these are only over a couple hundred entries at most, which is not a problem for C++ in terms of the speed.

Depth of inheritance:

Do the inheritance relationships between the ancestor/descendent classes go too deep in the hierarchy? (The deeper a class in the hierarchy, the greater the number of methods it will probably inherit from its ancestors, making it harder to predict its behaviour).

☐ Yes

☒ No

☐ Partly (Can be improved)

Comment on your analysis: There is no inheritance for the code in this requirement.

Comment on your findings: There is only one class that uses children classes and even with that it only goes 1 level deep.

Children:

Does a parent class have too many children classes? (This could possibly suggest an abstraction problem.)

☐ Yes

☒ No

☐ Partly (Can be improved)

Comment on your analysis: As before, there is no inheritance in the code for this requirement.

Comment on your findings: There is no inheritance not by lack of attention to design but because it would not be appropriate for these particular features.

Behavioural analysis:

From the system's requirements, create several scenarios starting from the user's point of view: consider identifying one or more typical scenarios (e.g., those expected to be used with high frequency) and one or more low-frequency scenarios .

Each scenario is described as follows:

- i) Title of scenario
- ii) Anticipated frequency of use (high, normal, low)
- iii) End-user trigger (starting point) for the scenario.
- iv) Expected type of outputs.
- v) List of bullet points linking end-user inputs and identifying all the key features of the system expected to be "touched" by the scenario and producing the anticipated outputs.

Follow the code (structured walkthrough) to ascertain whether this scenario is properly implemented both in terms of logic and design.

Comment on your findings, with specific references to the design/code elements/file names/etc.:

—

Title: View a line chart for the selected data.

Anticipated frequency of use: Relatively high frequency.

Starting point: Click on a name on the list, then click on the line chart button

Expected outputs: A line chart representing that data, which can be printed

Inputs:

- A chart list that contains the relevant information based on the current sorting option

Walkthrough:

- When the user clicks on a name on the list, it triggers a slot that begins creating graphs
- The various `treeView_total` methods create a list if the total option is selected
- Inline code creates a list by loading information from the tree models
- `SetupPieChart`, `setupBarChart`, `setupLineChart` and `setupScatterPlot` use this data to create the charts
- When a user clicks on an available chart type, and index is updated using `changeCurrentIndex` in QT which will update the current view appropriately

All the setup functions, namely `setupPieChart`, `setupBarChart`, `setupLineChart` and `setupScatterPlot`, do a good job of properly setting up the required graphical information. The code that creates the lists within the `treeView_clicked` block of code is not very versatile and could use reworking in future iterations of the project.

END.

cs3307a – Object oriented analysis and design

Design Inspection Instrument

Instructions:

The purpose of this document is to assist in the inspection of object-oriented design. Under each question is a choice of answers; please choose one (either replace the box with a checkmark or highlight it)

yes no partly, could be improved

Two types of comments are required under each question: (i) your analysis, and (ii) your finding (in the form of a comment). The analysis would typically show how you arrived at the finding.

Add new lines as necessary for your analysis or findings.

Scope and process:

Choose a subset of the system's features.

Create some scenarios in which these features will be used.

Keep the inspection process down to, say, two hours. This gives you some preliminary experience with inspection.

Log improvement suggestions.

Make improvements as appropriate.

+++++

Classes/Methods Inspected: MainWindow, selectData, QFileIO, and the various slots that are required to activate these methods

Performed by: Jeffrey Hsu

Structural correspondence between Design and Code:

Are all the classes and interrelationships programmed in the application explicitly represented in the class diagram of the system?

Yes No Partly (Can be improved)

Comment on your analysis: A diagram was generated using another diagram software to aid with this. By inspection, all classes in the codebase are included in the diagram

Comment on your findings: All the Classes were found to be within the class diagram along with the corresponding relationships.

Functionality:

Do all the programmed classes perform their intended operations as per the requirements?

Yes No **Partly (Can be improved)**

Comment on your analysis: Yes. See the selectData and QFileIO test documentation for further details. on_teachCustomList_clicked, on_pubCustomList_clicked, on_presCustomList_clicked, and on_fundCustomList_clicked all call methods to bring up a window of buttons that invoke the methods in the selectData class such as getCustomFields and displayNames. savePath and readPath call methods that take a parameter string and serializes or reads from a serialized file the path where the .csv file is stored

Comment on your findings: Custom lists must be made through a separate window rather than double clicking which is less than optimal but it works as the customer described. QFileIO does not really save the session but just the path of the file that was opened previously and Mainwindow takes those paths and loads the files at the end of them

Cohesion:

Do the methods encapsulated in each programmed class, together perform a single, well defined, task of the class? (High-Cohesion (good): the functionalities embedded in a class, accessed through its methods, have much in common, e.g., access common data)

Yes No **Partly (Can be increased)**

Comment on your analysis: selectData only accesses the strings storing member names in the Mainwindow databases and the custom databases are setup inside Mainwindow. QFileIO accesses the strings storing the file paths and serializes them to a file, no modifying is done

Comment on your findings: selectData and QFileIO do not modify original information but modifies a copy of it and Mainwindow uses that copy only temporarily, ie. all original data remains unchanged

Coupling:

Do the programmed classes have excessive inter-dependency? (High Coupling (bad): In this case a class shares a common variable with another class, or relies on, or controls the execution of, another class.)

Yes No Partly (Can be reduced)

Comment on your analysis: No common variables with other classes, obviously relies on the Custom List button in Mainwindow being clicked to execute out of necessity, and QFileIO relies on a file actually being imported into the program, but that's all

Comment on your findings: no other classes can access variables in selectData or QFileIO

Separation of concerns:

Is the scoped problem decomposed into separate concerns where each concern is encapsulated in a construct such as a class with well-defined interface and cohesive functions with minimal of connections with other concerns?

Yes No Partly (Can be improved)

Comment on your analysis: selectData handles the concern of selecting out the desired names which the user wants to display, Mainwindow takes that list and gets the information pertaining to those names from its database and stores it in a new database. QFileIO handles the concern of storing knowledge of a previously opened file and it is used by Mainwindow to locate such files

Comment on your findings: Each of the data tabs required a different way to search for and retrieve specific data so they had different solutions using the selectData class

Do the classes contain proper access specifications (e.g.: public and private methods)?

Yes No Partly (Can be improved)

Comment on your analysis: Extensive look through where any class was given new methods along with new class that was implemented

Comment on your findings: Found that each new method and class implemented entailed the proper access specifications based on how and who would call it.

Reusability:

Are the programmed classes reusable in other applications or situations?

Yes, most of the classes No, none of the classes Partly, some of the classes Don't know

Comment on your analysis: selectData could be compatible with strings beyond Member Name, with a bit of tweaking it could make custom lists of any string. QFileIO, can be extended to saving the paths of any type of file so long as MainWindow has a function available for loading in said type of file

Comment on your findings: the super class MainWindow is reusable as it properly controls what is needed to be executed based on the user selections. selectData and QFileIO have generalized code which can be extended to accommodate more features with minor adjustments

Simplicity:

Are the functionalities carried out by the classes easily identifiable and understandable?

Yes No Partly (Can be improved)

Comment on your analysis: Discussed among team-members what they thought about the code when each of them individually analyzed specific parts of the code before they implemented their tasks and requirements.

Comment on your findings: Found that although it can be understood what the system and certain classes do on a whole, examining specific methods require a lot more time as there is barely any comments indicating what certain lines do.

Do the complicated portions of the code have comments for ease of understanding?

Yes No Partly (Can be improved)

Comment on your analysis: Once again discussed as a team what individuals thought about the code when they examined it in depth.

Comment on your findings: Found a noticeable lack of comments throughout most classes. Particularly large methods, that computed a number of tasks, were given the least amount of attention when explaining certain computations. Because of this, the team spent a vast majority of time trying to understand what certain methods did in order to implement their own. This was especially time consuming when bugs would come up and there was no real way to tell, in which line/function, this was being caused.

Maintainability:

Does the application provide scope for easy enhancement or updates? (e.g., enhancement in the code is not anticipated to require too many changes in the original code)

Yes No **Partly (Can be improved)** Don't know

Comment on your analysis: methods such as checkFile were extremely difficult and time consuming to debug and to add a simple enhancement would require changing code and parameters of several different methods

Comment on your findings: In regards to this project, where enhancements were the main portion of this assignment, we found that big enhancements and features were not easy to implement. Once again, majority of time is to be used in understanding all the classes/methods which we determined not to be easy. On top of this, there is very little flexibility in regards to these classes as they take in very specific parameters and are not easy to manipulate.

Efficiency:

Does the design introduce inefficiency in code (e.g., causes too many nested loops or delays in concurrent processing)?

Yes No **Partly (Can be improved)** Don't know

Comment on your analysis: the displayNames method in selectData likely has more loops than it requires and is overall rather unoptimized but the time and space complexity of the feature itself is small enough that any delay is negligible

Comment on your findings: there were a lot of loops and if statements but no actual visible delays when executing the Custom List feature

Depth of inheritance:

Do the inheritance relationships between the ancestor/descendent classes go too deep in the hierarchy? (The deeper a class in the hierarchy, the greater the number of methods it will probably inherit from its ancestors, making it harder to predict its behaviour).

Yes **No** Partly (Can be improved)

Comment on your analysis: There is no inheritance for the code in this requirement.

Comment on your findings: There is only one class that uses children classes and even with that it only goes 1 level deep.

Children:

Does a parent class have too many children classes? (This could possible suggest an abstraction problem.)

Yes

No

Partly (Can be improved)

Comment on your analysis: Comment on your analysis: As before, there is no inheritance in the code for this requirement.

Comment on your findings: There is no inheritance not by lack of attention to design but because it would not be appropriate for these particular features.

Behavioural analysis:

From the system's requirements, **create several scenarios** starting from the **user's** point of view: consider identifying one or more **typical** scenarios (e.g., those expected to be used with high frequency) and one or more **low-frequency** scenarios .

Create a custom list [high frequency]

- Click Custom List in the main window

- Custom List window will pop up

- Double click desired member names

- Names that are double clicked will move from the left window to the right window

- Double click member names in the right window to remove or click Reset to start over

- Otherwise click Ok if satisfied with list

- Custom List displays in the main window

Comment on your findings, with specific references to the design/code elements/file names/etc.: Custom List works as per the customer specification but it is done through a separate window which may be less convenient than implementing a control+click feature. Save session state suffices for the current functionality of the program but will have to be looked at in the future if a feature to save edited errors were to be implemented.

END.

Implementation

To reiterate, both the design patterns of the singleton and prototype were used for the phase 1 system and continued through the phase 2 implementation. The singleton is still prominent in that a single `MainWindow` is created and this object controls the runtime behaviour of the graphical user interface. This design decision is a result of the belief that there must be exactly one instance of this class and it must be accessible to clients from a well-known access point. New methods were added to other classes for new calculations required for the deliverables. Because not much has changed to the actual `MainWindow` we can safely say that the singleton pattern is still very much the backbone of this program.

In the case of `TreeModel`, we use what is referred to as a prototype manager. When the number of prototypes in a system isn't fixed (that is, they can be created and destroyed dynamically), we keep a registry of available prototypes. Clients will not manage prototypes themselves but will store and retrieve them from the registry. A client will ask the registry for a prototype before cloning it. Unfortunately our prototype classes do not define operations for (re)setting key pieces of state. If not, then you may have to introduce an initialization operation that takes initialization parameters as arguments and sets the clone's internal state accordingly. An example of this is the `setupModel()` operation in `Treeltem`. In terms of the new classes created for Demo 2, both "editSort.cpp" & "selectData.cpp" these, although small and handle specific data, more or less follow the prototype pattern in that it follows other smaller classes controlled by `MainWindow` so that clones or copies can be created.

In the case of the graph types implemented as per mandatory requirement #2, this was designed to be an extension of the user interface implemented inside the `MainWindow` class. This is because QT requires all of the slots attached to UI elements to be implemented in a file with the same name as the UI form. Because of this design, all of the methods implemented to enable this functionality were included inside this class. This includes the implementation of `setupLineChart` and `setupScatterPlot`, as well as all the helper methods that setup the charts with the required data, such as `teachTreeView_total`.

The team of Orion found no need to stray away from the current patterns used as a backbone for this program. We follow these designs as closely as possible, making certain that any new code added does not become more confusing to read than what is originally there.

Development Plans

Stage 1 Timeline

Team Orion - Peach Galaxy Phase 2						
					TimeLine	
					Deliverable 3 sprint	
Achievements					Estimated Due Date	Date completed
Phase 1 system's Documentation analyzed					October 16, 2016	October 16, 2016
Phase 1 sustem's Documentation completion					October 20, 2016	October 21, 2016
Improvements to phase 1 system identified and logged					October 18, 2016	October 18, 2016
Test Cases created for Phase 1 system					October 20, 2016	October 21, 2016
Demo 1 requirements determined					October 27, 2016	October 27, 2016
Demo 1 requirements tasks allocated to members					October 27, 2016	October 27, 2016
Development infrastructure set up					October 29, 2016	October 30, 2016
Requirements completed					October 30, 2016	October 31, 2016
Test-cases completed					October 31, 2016	October 31, 2016
Documentation of deliverable 3					October 31, 2016	October 31, 2016

Plan that was demonstrated during the sprint of Deliverable 3. Starting dates for the achievements were not documented but it is to be noted that the requirements involving the stage 1 program begun initiation on the 20th of October.

Stage 2 Timeline

Team Orion - Peach Galaxy Phase 2						
			TimeLine			
			Deliverable 4 sprint			
Achievements				Date Started	Estimated Due Date	Date completed
Demo 2 requirements determined				October 31, 2016	October 31, 2016	October 31, 2016
Demo 2 requirements allocated to members				November 3, 2016	November 3, 2016	November 3, 2016
Demo 2 requirements completed						
M.R.2				November 3, 2016	November 14, 2016	November 18, 2016
M.R.5				November 3, 2016	November 14, 2016	November 21, 2016
M.R.9				November 3, 2016	November 14, 2016	-
Demo 2 Test-cases completed				November 15, 2016	November 23, 2016	-
System Diagrams completed				November 23, 2016	November 23, 2016	November 23, 2016
Documentation of deliverable 4				November 23, 2016	November 23, 2016	November 23, 2016

Final Stage Timeline

Team Orion - Peach Galaxy Phase 2						
			TimeLine			
			Deliverable 5 sprint			
Achievements				Date Started	Estimated Due Date	Date completed
Fix bug for new spreadsheets being entered				November 20, 2016	November 30, 2016	December 2, 2016
Demo 2 requirement of M.R.9				November 3, 2016	November 14, 2016	December 4, 2016
Test cases for M.R.5				November 20, 2016	November 30, 2016	December 5, 2016
Test cases for S.R.2				November 20, 2016	December 3, 2016	December 5, 2016
Installation package Created				December 5, 2016	December 7, 2016	December 7, 2016
System Diagrams completed				December 4, 2016	December 6, 2016	December 7, 2016
Documentation of deliverable 5				December 3, 2016	December 7, 2016	December 7, 2016

Agent-Task View

Stage 1 Task Allocations

Tasks	Paul	Chris	Yuchen	James	Jeff	Ming
Deliverable 3						
M.R.2 - 1		x	x			
M.R.3	x				x	
M.R.4				x		x
Test M.R.2		x	x			
Test M.R.3					x	
Test M.R.4				x		x
Class Diagram v1				x		
Use-Case Diagram v1		x	x			
Development Plans	x					

Stage 2 Task Allocation

Deliverable 4	Paul	Chris	Yuchen	James	Jeff	Ming
Bug1	x			x		
M.R.2 - 2		x				
M.R.5					x	x
S.R.2			x			
Test-M.R.2		x	x			
Use-Case Diagram v2		x				
Class Diagram v2				x		
Sequence Diagram v1	x					
Package Diagram v1						x
Design Patterns	x					
Implementation in C++	x	x		x	x	x
Development plans	x					

Final Stage Task Allocation

Final Stage	Paul	Chris	Yuchen	James	Jeff	Ming
Bug1	x			x		
M.R.9	x			x		
M.R.6	x	x	x	x	x	x
M.R.7	x	x	x	x	x	x
M.R.8					x	
S.R.2			x			
S.R.3	x	x			x	x
Test-M.R.9	x			x		x
Test-M.R.5					x	x
Test-Final	x	x	x	x	x	x
Use-Case Diagram v3		x				
Class Diagram v3				x		
Sequence Diagram v2	x					
Package Diagram v2						x
Design Patterns	x					
Implementation in C++	x	x	x	x	x	x
Development plans	x					

Lessons Learnt

Top Three Lesson Learnt (Technical issues)

1) The team suspects that one of the main reasons that this project was assigned was to give students experience in maintaining and enhancing code that was written by others. Many programmers experience this challenge more often than they start a project from scratch. A lot of time was spent debugging and going through the code, sometimes line by line. When it comes to this kind of work, patience is always a good thing to keep in mind and to be optimistic that as a team member you will find the problem.

2) The team learned how to utilize new software and IDEs to help accommodate this project. Tools such as text editors, repositories, video editors and IDEs (eg. Qt Creator). The main software learned would be that of Qt. Qt is an IDE for C++ code that can also be utilized as a development tool for user-interfaces. We Learned how to add forms and buttons and that adding layouts to pages often requires a series of unintuitive steps. Frustrations aside, QT allows for the creation of powerful user interfaces.

3) Even though we had performed an initial analysis to understand the first phase of the project, we did not have a full appreciation for the complexity of the code until we started adding features. At times it felt as if we should just refactor the entire project ourselves because we felt limited by the design decisions of the members of Team Peach. Ultimately, in the interest of balancing time, we chose to retain most of the Team Peach code base and added our own features. But we found it tricky to integrate the code, which caused bugs to arise that Team Peach would not have encountered in their limited testing of the project.

Retrospective Analysis (Human issues)

Analysis of Team organization: Individual roles and tasks were well defined. The project proceeded at a reasonable pace and significant progress was made for each deliverable. All team members were able to communicate effectively and responded to messages from other team members promptly. The team made effective use of github for version control and was able to collaborate productively to resolve issues.

Management: Management aspect of the project was implemented in a professional manner. Scheduling meetings and group work according to everyone's liking with the least amount of conflicts possible. Luckily for the team no conflicts arose in terms of system design, scheduling or group work. Project Manager would allow members to choose which activities they wanted based on their preferences that way members could have some passion in what they were doing instead of being given something they would not have liked. Members were also partnered up so they would have someone in the group to go to for help instead of messaging the entire team. If a huge conflict would arise in terms of merging or helping a team-mate, then the group would arrive together and solve the problem collectively instead on separately.

Project Performance: Project was scheduled so that different requirements can be completed before others that may otherwise interfere with them. Requirements were initiated and testing accordingly before new requirements began. All members were utilized accordingly to not only what their strengths were but also what they, as a person, wanted to do. Because of this members performed to the best of their ability.

Value

- We developed an appreciation for just how significant documentation is in any software development project. It can be just as significant as the coding portion, and from a user's perspective, is even more so.
- We learned about different kinds of diagrams (like Use Case, Sequence and Package diagrams) that are useful for users and future maintainers of the project when trying to understand the design of the Orion Galaxy project.
- We learned about different design patterns and how they can be useful in designing and developing a larger codebase.
- This was the first time that many of the group members have used C++, so the team learned the syntax of the language and how to utilize its object oriented features to more easily develop and maintain a large project.
- We learned about version control with GitHub and the importance of communicating any modifications to the program to avoid conflicts and possibly breaking the program.
- We learned how to communicate effectively with the customer to ensure that what they want is properly implemented to the best of our abilities.
- Our group specifically learned to consolidate and keep progressing on the back of a member having to drop the course.
- The group had to react to changing requirements from a customer (with the format of the sample data files changing) in a short amount of time. This caused issues throughout the program.