CS3307 – Object Oriented Design and Analysis
Fall 2015

Dashboard Project

Team Grapefruit

James Crocker, Colin Costello, Eric Lefebvre, Gao Song, Junwon Seo, Peter Pfoertsch, Yiming Guan, Kevin Tawaststjerna, Lankesh Patel, Larsen Burchall

Table of Contents

Minimum Requirements

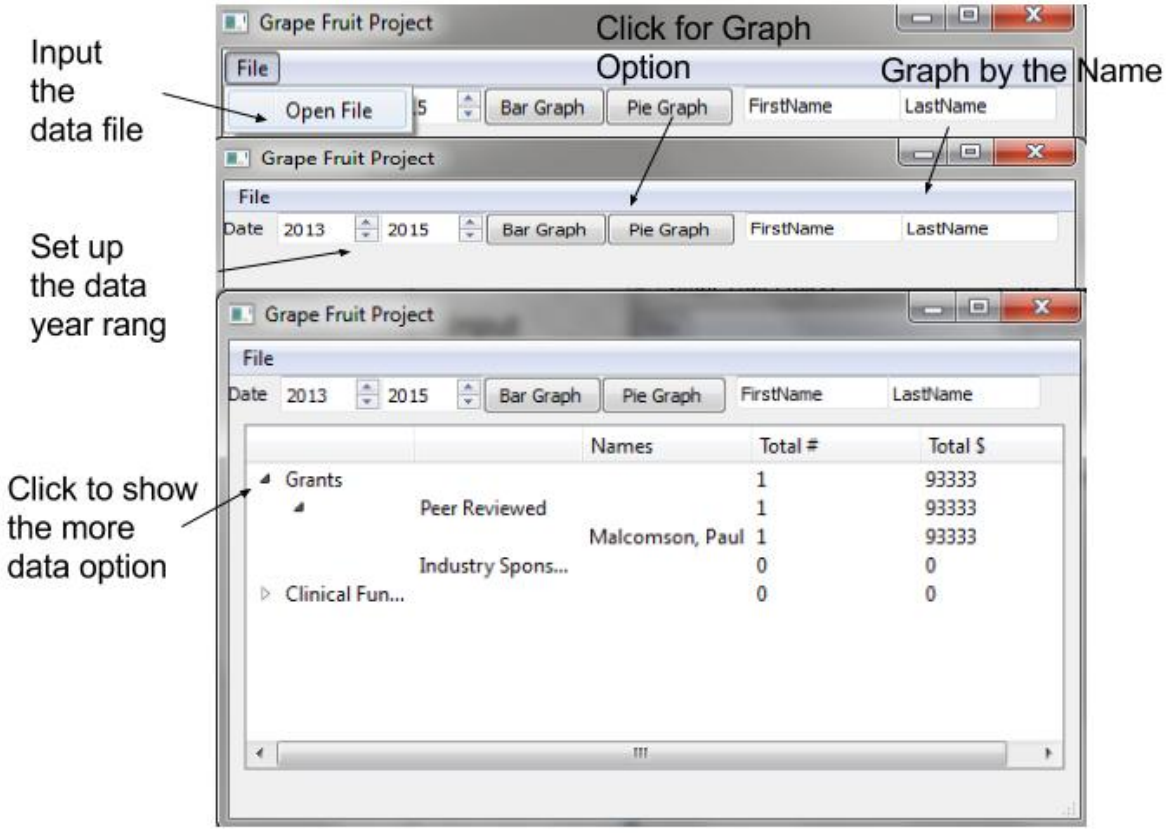
This project is build for the purpose of data collection and basic graphic analysis. The minimum requirements specified the following for the Teaching, Grants, Publication and Presentation ".csv" files:

- Display of the data in an expandable +/- list
- Display data in bar graph
- Display data in pie graph
- Change date range
- Print visualizations

The following section provides evidence of the implementation of the minimum requirements for this project.

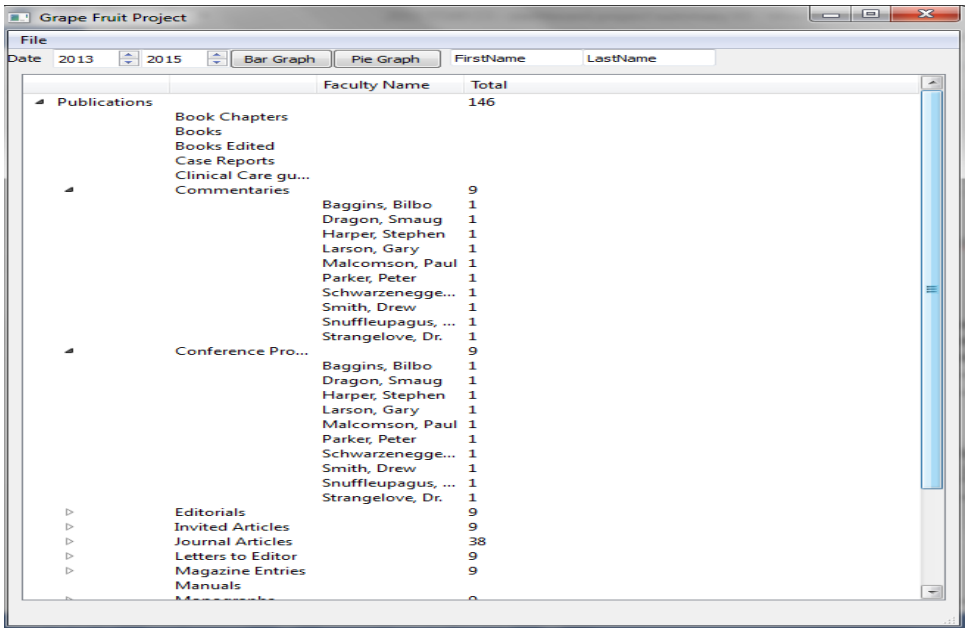
Data Visualization

The basic features of the program are identified in the image below:

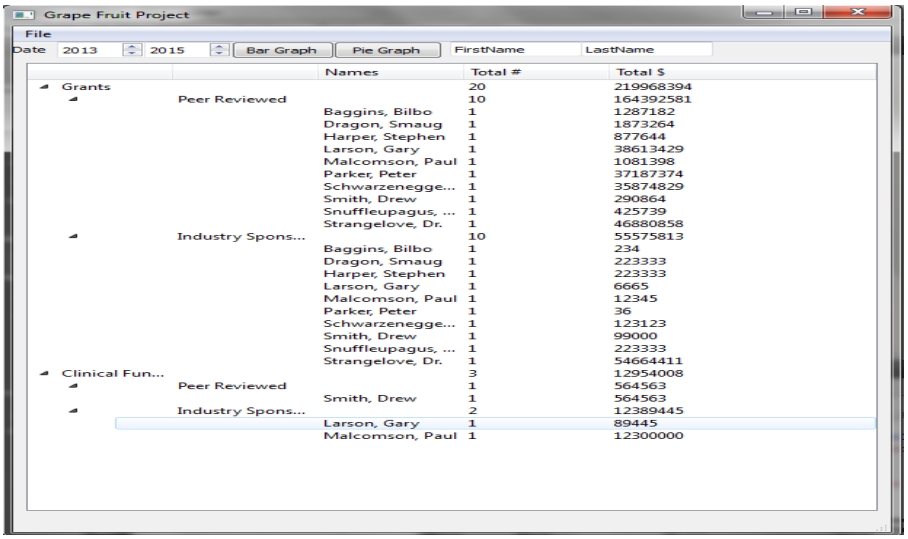


The following portion of this section demonstrates the Publications, Grants, Teaching and Presentation ".csv" data being displayed in expandable +/- lists.

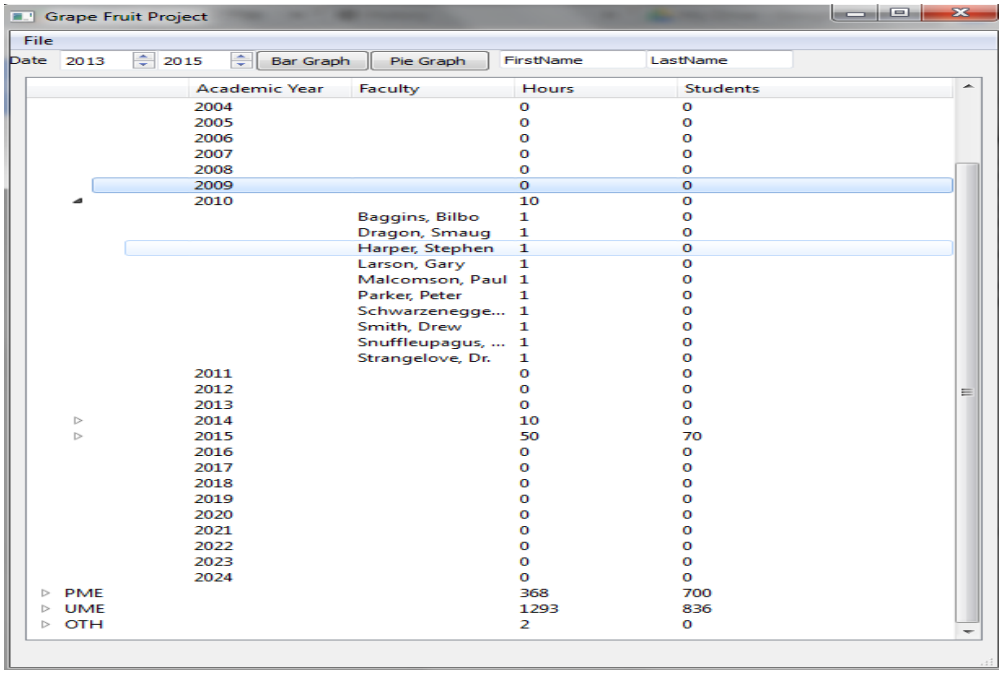
Below is a visualization for Publications data. All the Publication data is classified by one of the 20 publication's categories. For each classification, the faculty member's name and number of publications are displayed.



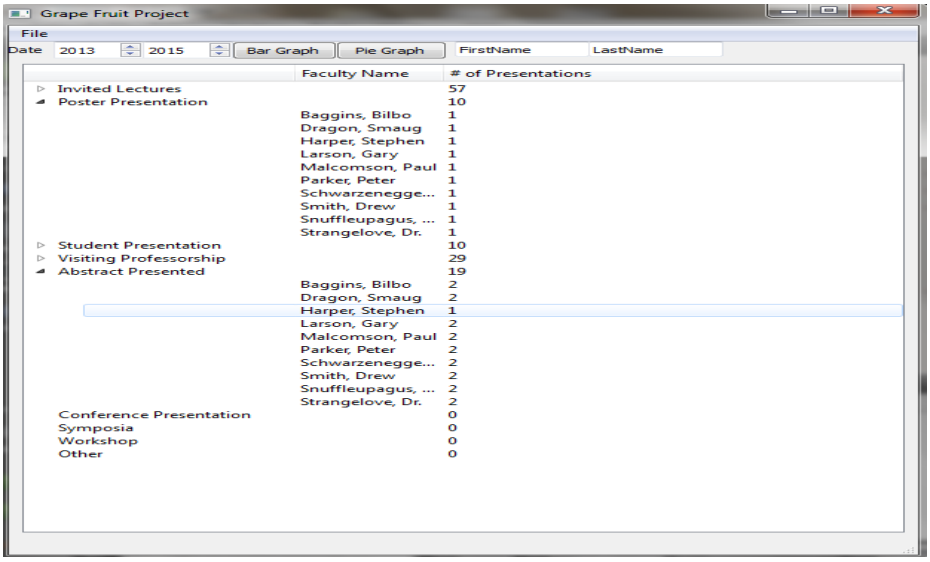
Below is a visualization for Grants and Clinical Funding data. All the data is classified by Grants and Clinical Funding, and subsequently whether the point has been Peer Reviewed or Industry Sponsored. For each inner options, the faculty member's name and funding amount are displayed.



Below is a visualization for teaching data. It is classified by one of the four teaching levels (e.g PME, UME, CME etc). For each level of teaching, it contains faculty member's name, number of teaching hours, and number of students taught.



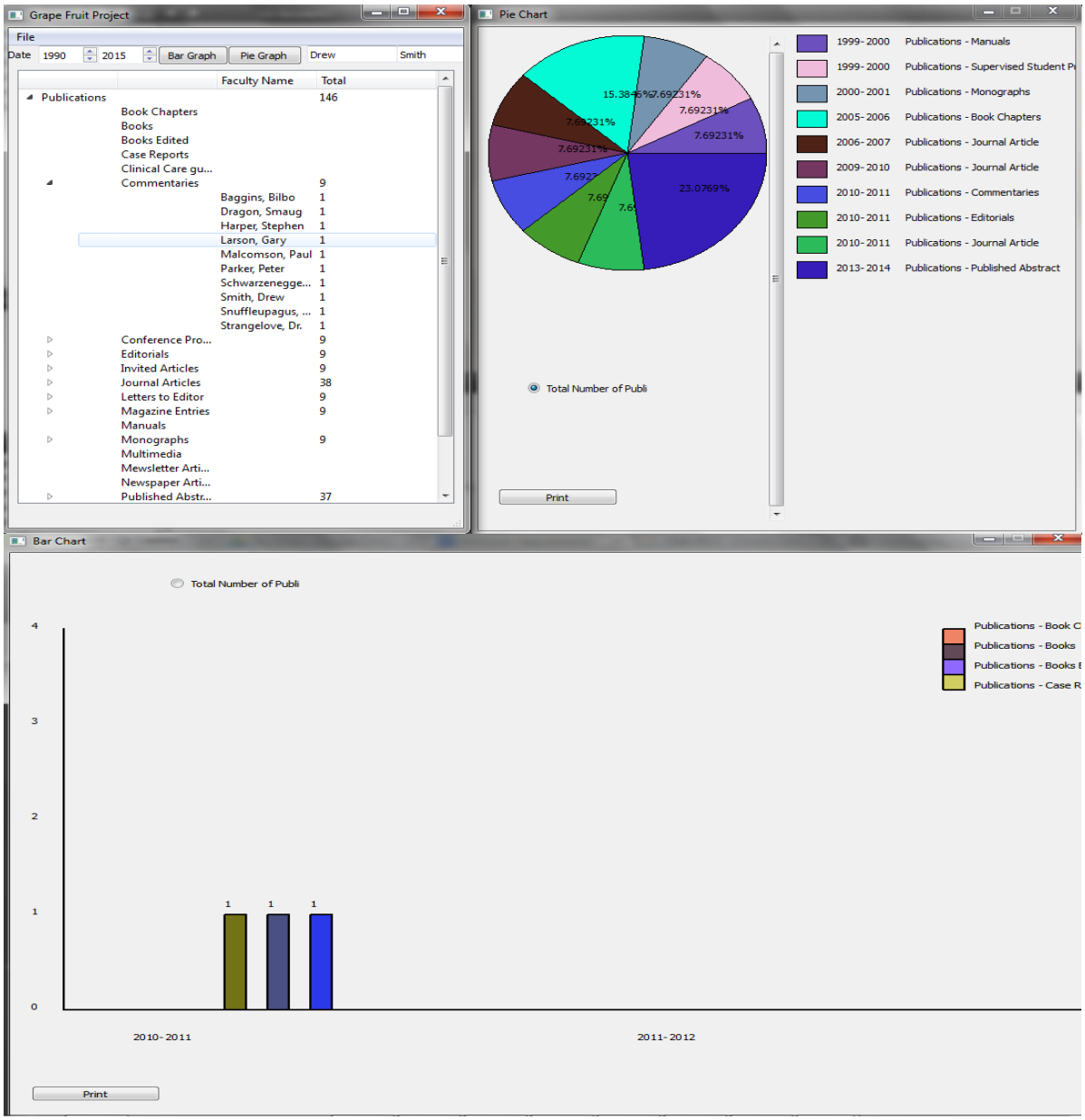
Below is a visualization for Presentations data. This data is first classified by the presentation type, followed by the faculty member. Along with the faculty member's name, the number of presentations that they presented are displayed.



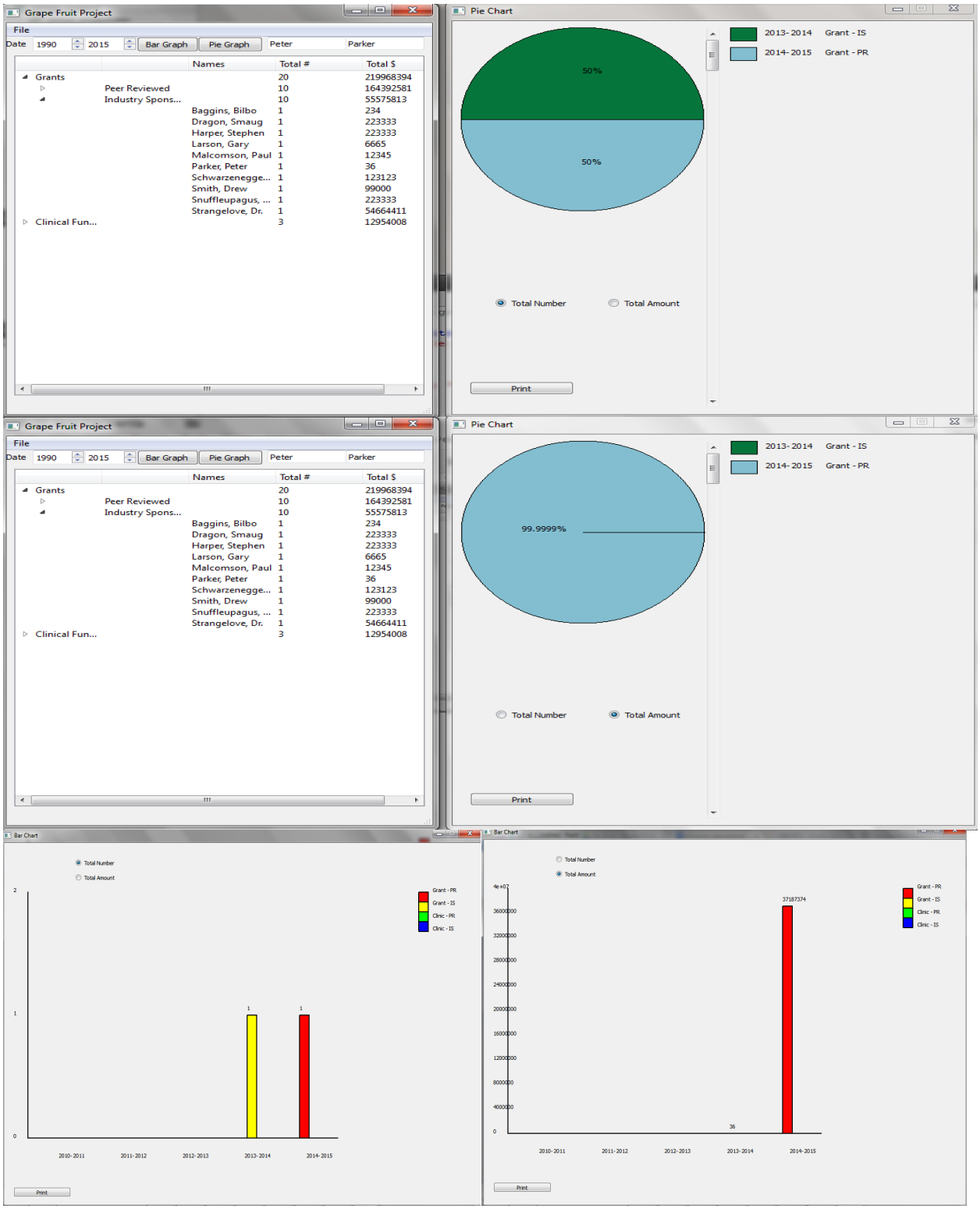
Graph Visualization

The following portion of this report demonstrates the functionality of the bar and pie graphs for each of the four ".csv" file types.

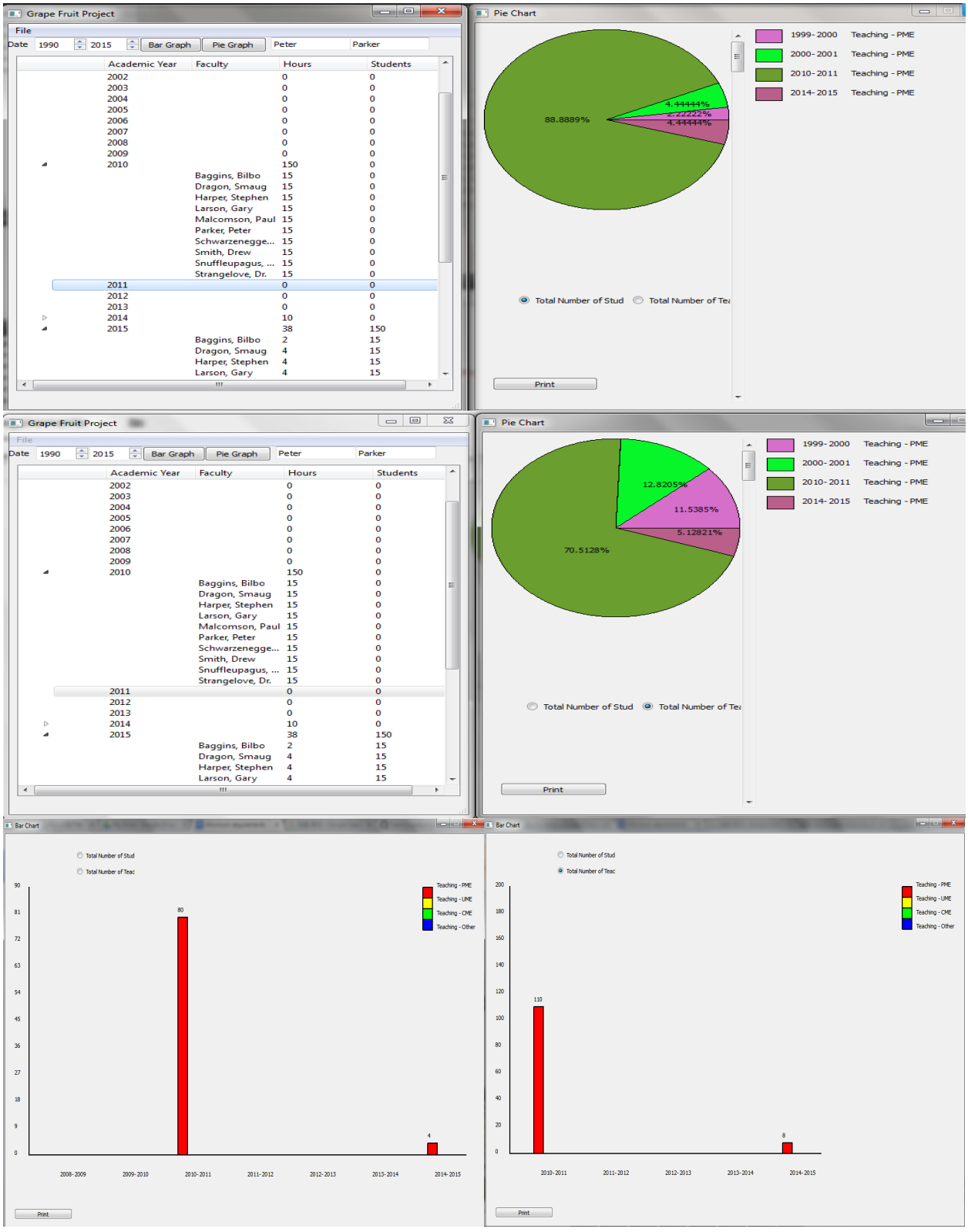
The Publication's graphs can be seen below for faculty member "Drew Smith" from 1990 to 2015. For ease of reading, only those years where a publication was completed are displayed. For example, 1990 to 2010 do not appear on the x-axis because Drew Smith did not release a publication in this time frame.



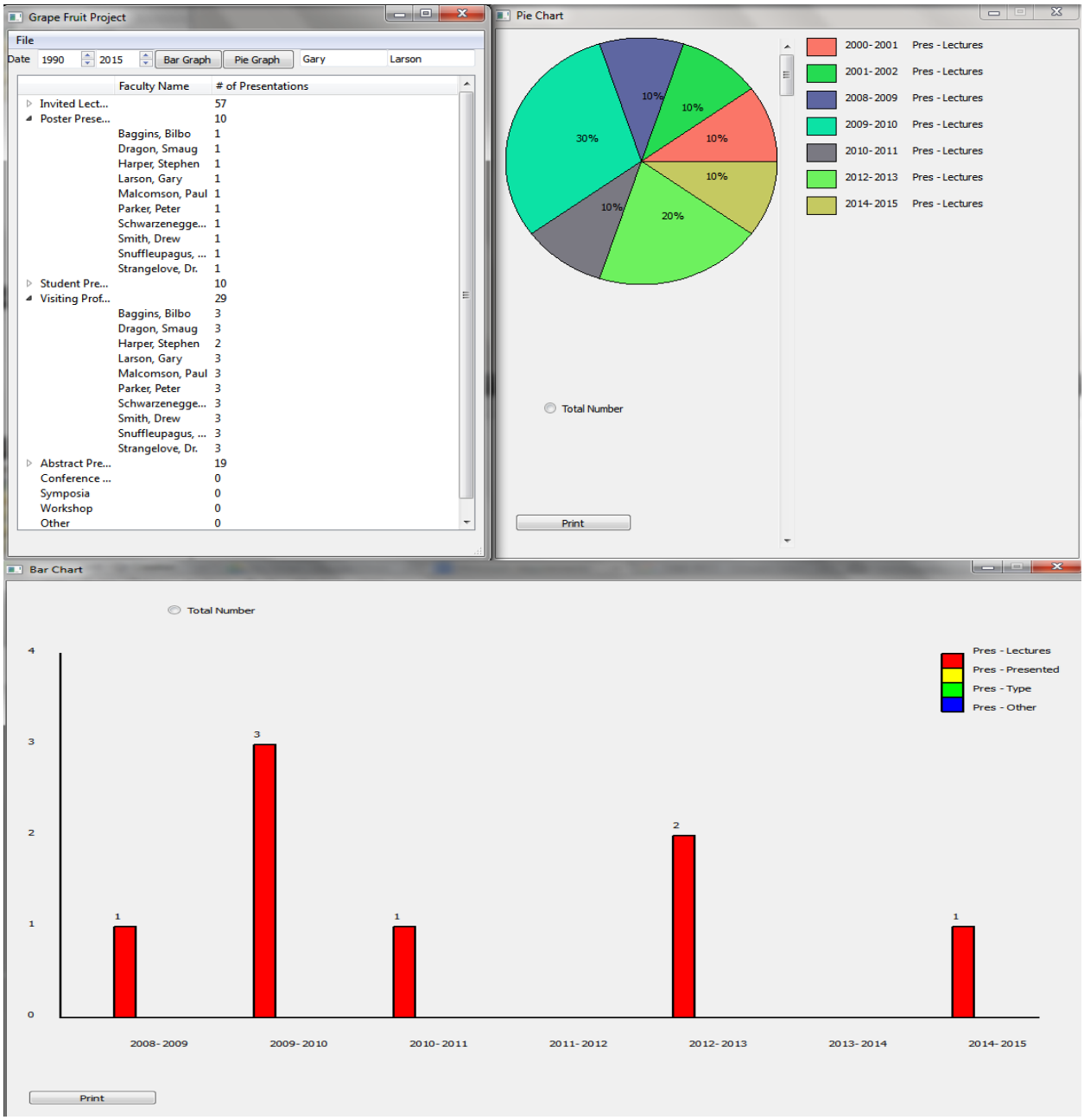
An example of the Grants graphs can be seen below. This data is displayed for faculty member “Peter Parker” from 1990 to 2015.



An example of the Teaching graphs is displayed below. This data is displayed for faculty member “Peter Parker” from 1990 to 2015.



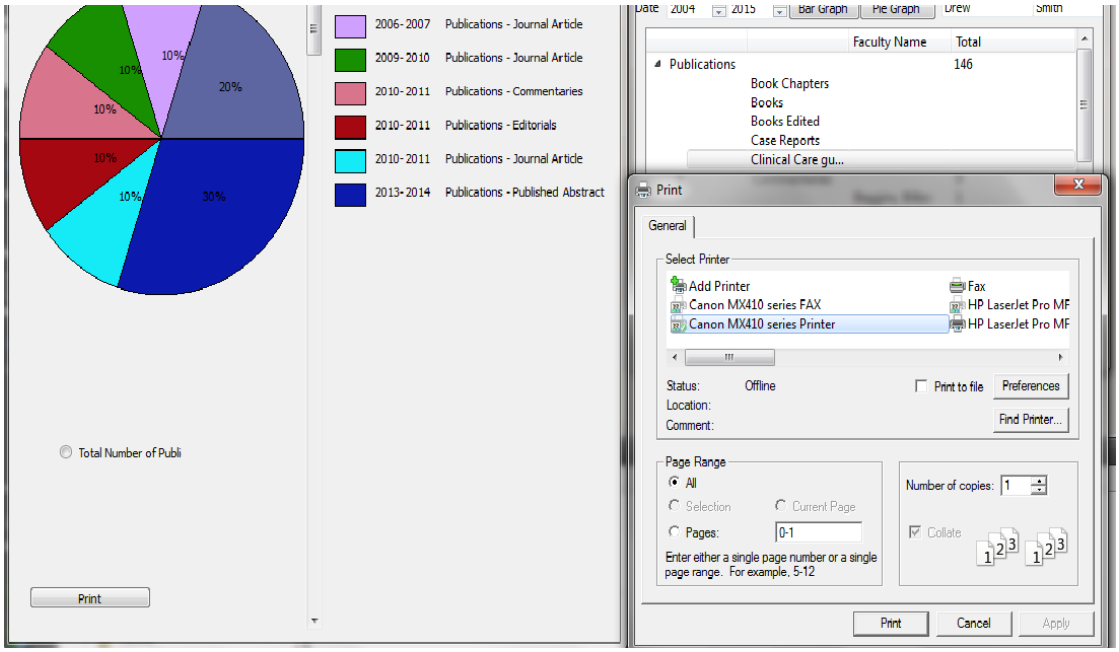
Finally, an example of the Presentation graphs is displayed below. This data is displayed for faculty member “Gary Larson” from 1990 to 2015.

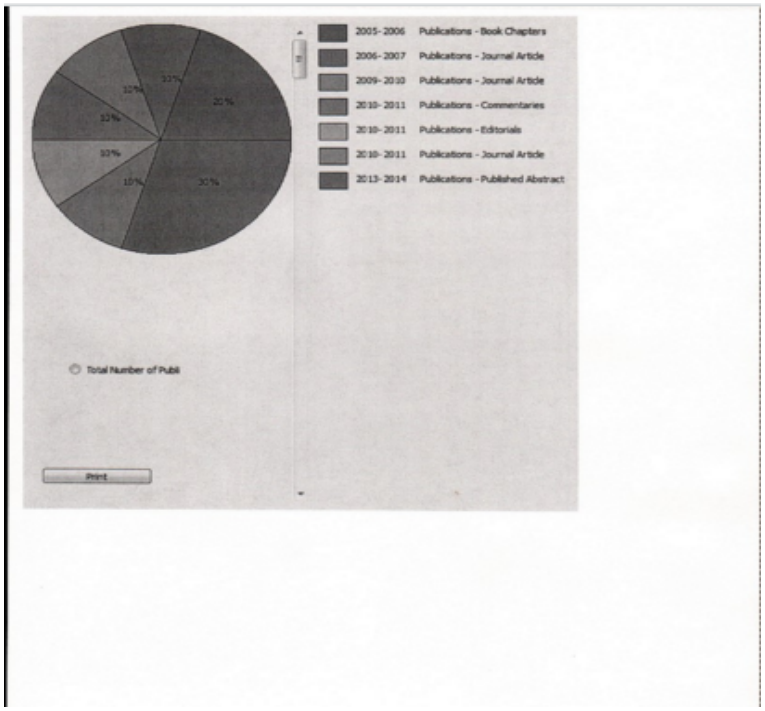


Graph Printing

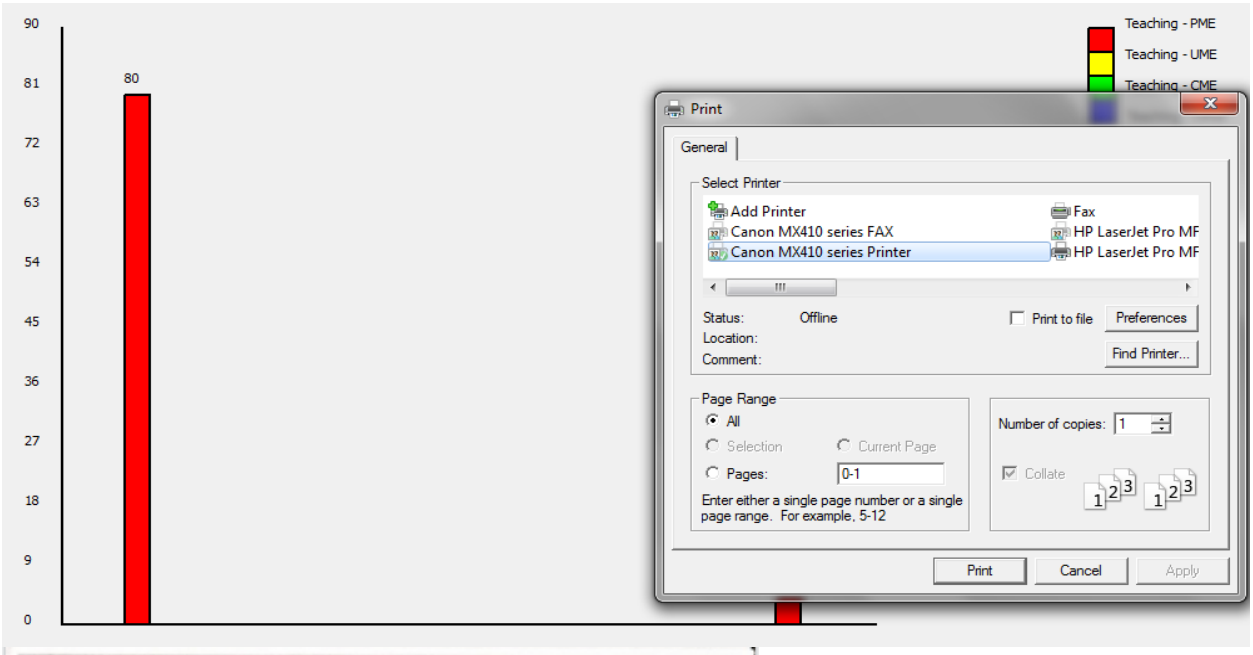
The following portion of this report demonstrates the functionality of printing for input ".csv" file types.

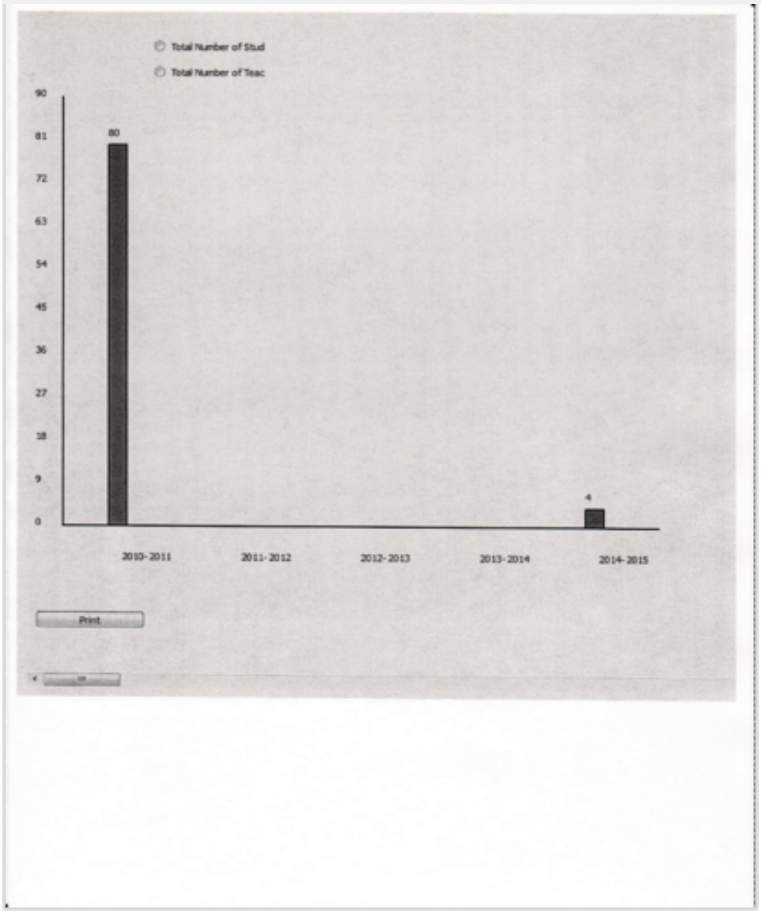
This is an example of Pie Chart printing Publications 2004-2015.





This is an example of Bar Chart printing for Teaching 2010-2015.





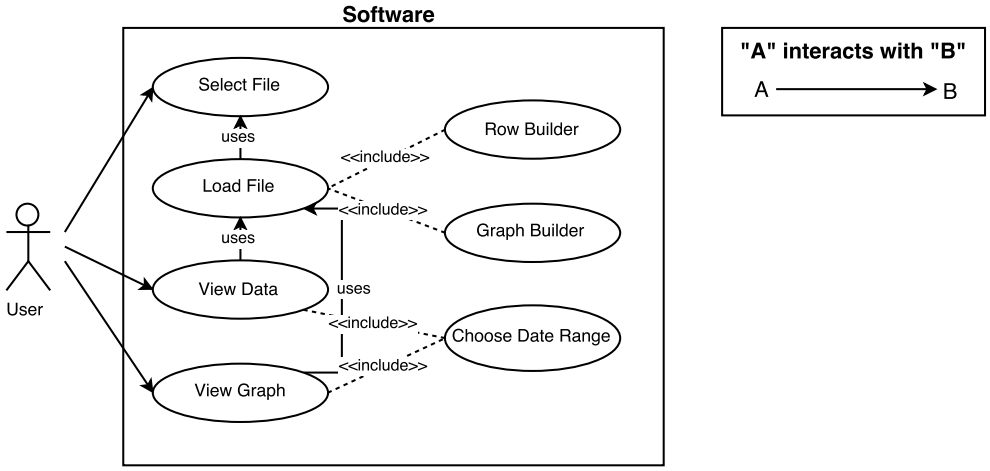
Stretch Requirements

The team has implemented the required error checking; if required text fields are left blank, or if date fields are left blank/have a value of zero, the program will add these rowObjects to an ‘Error Queue’. The team has extended this error checking to certain text fields that have a finite number of options (i.e.: the status of a publication can only be one of: "Published", "In-Press", or "Accepted"). Some of these fields are mandatory, thus the team decided that it would be worth throwing an error if the entry in these fields do not conform to one of its allowable values. This is implemented using the stringAssert(...) function in the ErrorChecker class.

All error caught were (a) added to an ‘Error Queue’ and (b) the entry from which the error was found was changed to an obviously wrong entry (i.e.e.: "**ERROR_BLANK_FIELD**"). In the user interface, there is a button displaying the number of errors. when the user clicks on this button, a list of errors pops up. This list indexes rows that have generated an error to the rowObject generated. Since erroneous rowObjects will have one or more field with an obviously wrong entry (see above), the user will easily be able to tell where in the CSV file the error is, and will thus be able to fix the errors by modifying the CSV file.

System Design

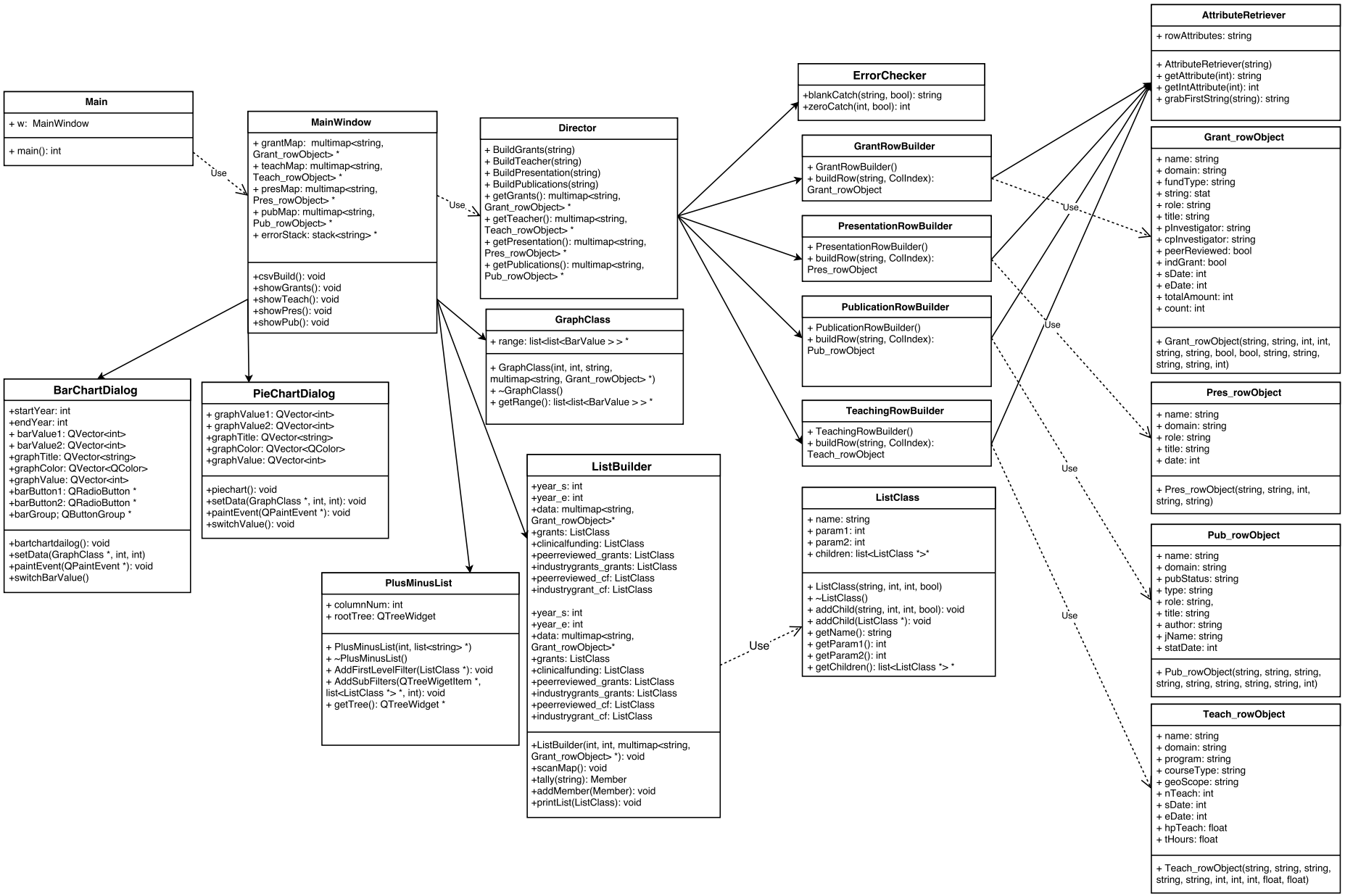
Use Case Diagram



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

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

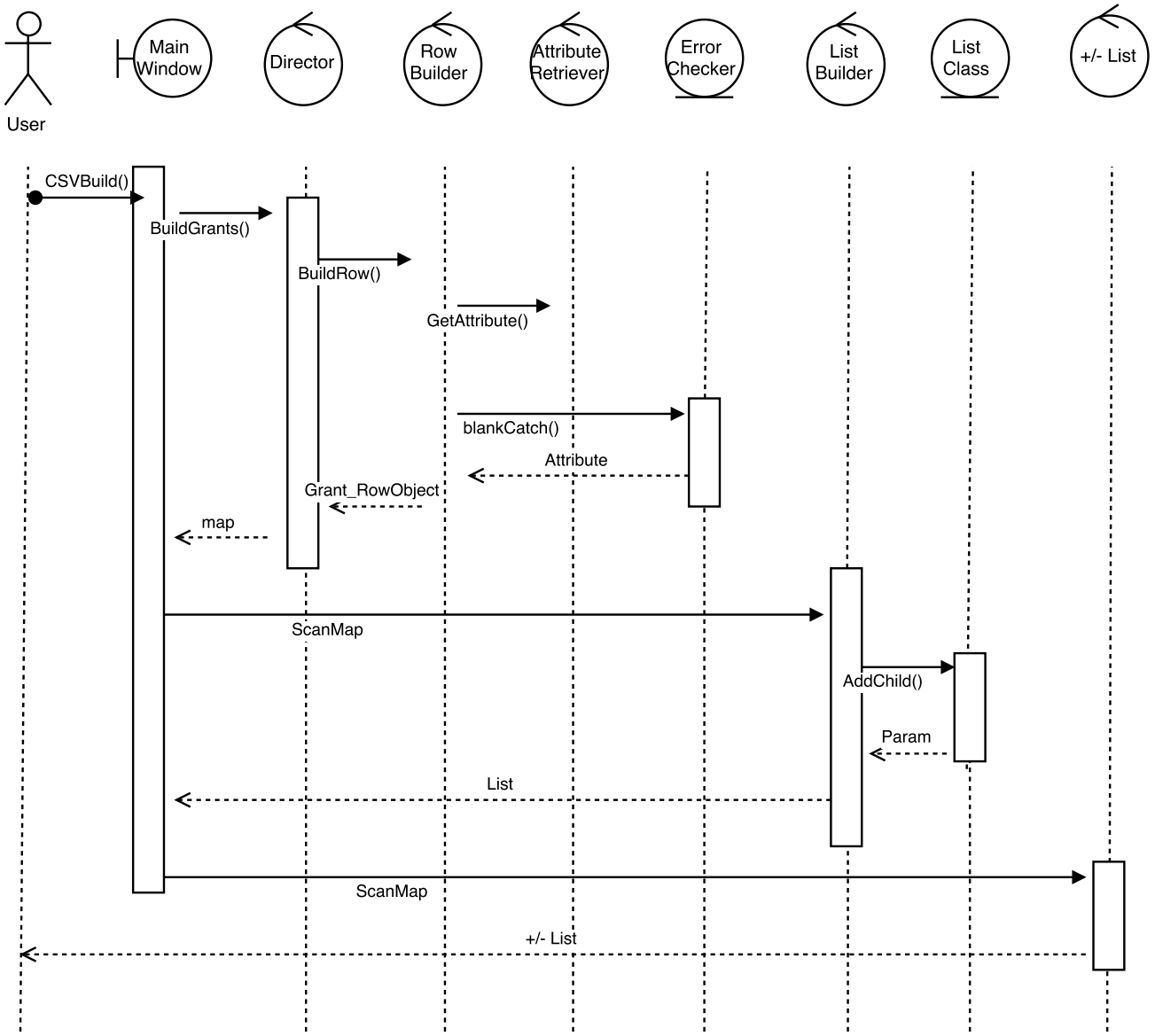
Class Diagram



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

2) The design for this program was based on the Builder Design developed by The Gang of Four. This allowed use to break the tasks of file parsing and user interface into levels. On the file parsing side, the Director class "pulls the strings" of building row objects from a high level. The Director will parse the ".csv" file and pass each parsed row to a RowBuilder class, which is at a level below it. The RowBuilder is responsible for building the row object by calling the AttributeReceiver which performs one level lower operations by retrieving a specific attribute from a given row. For example, the AttributeRetreiver will remove a name, date, string or integer from a ".csv" row and return it to the RowBuilder so that it can be assigned to a row object member variable. From here, the row object is passed back to the Director and inserted into a map. The creation of this map allowed for simpler processing down the line to form objects which could be passed to a generic graphing tool, or generic +/- expandable list tool. The benefit of this strategy was regardless of whichever of the 4 file types being used, one GUI class (either for graphs or the +/- expandable lists) could display all of them without having specific implementation for each file type. At the implementation level, this was done by the ListBuilder which creates a ListClass object from a row object map and the GraphClass which creates a List<List<BarValue > > *. These two objects are then passed to the PlusMinusList and graph dialog classes respectively for display in the MainWindow.

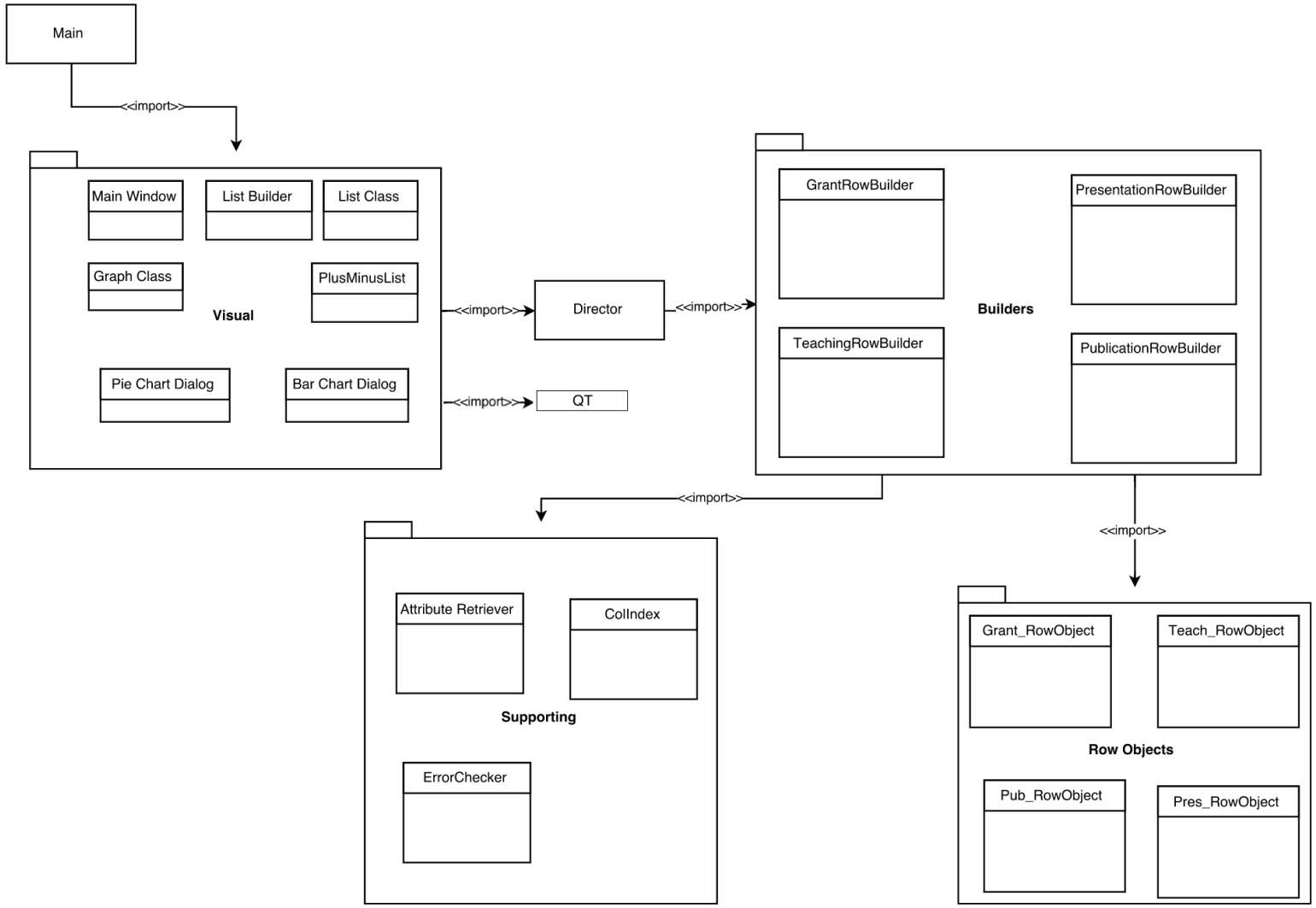
Sequence Diagram for a Scenario



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

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

Package Diagram

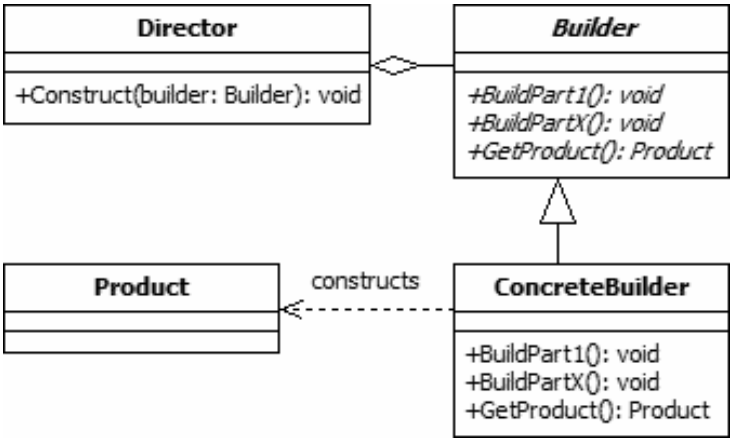


1) The diagram conveys the classes by grouping them based on their function towards the overall program. Every class falls under a grouping (or package), except for the Main and the Director classes, which stand separately. Arrows connect the packages and portray the hierarchy of the implementation.

2) The rational for the design of this diagram was to group the classes as best as possible by their function in relation to the overall program. The Visual package entails all classes that are specifically used for the user interface aspect of the application. The Builder package and Row Object package encompass the four builders and four row objects respectively, with each builder and row object corresponding to one of the four types of files. The Supporting package contains the classes that are used in association with the other packages, but don't relate to the other classes enough to be put in the same package. The Main and Director classes stand separately as they do not share the same functionalities as the other packages, and are not Supporting classes either.

Design Patterns

The design pattern we have implemented is the Builder Pattern. It is a Gang of Four design pattern that is used to create complex objects with constituent parts that must be created using a specific algorithm. An external class, the director, controls the algorithm. An example of this UML diagram can be seen below.



The builder class defines all of the steps that must be taken to correctly create a product. The director class controls the algorithm that generates the final product object. It calls the methods of the concrete builder in the specific order necessary to generate the desire product object.

Our implementation of the Builder Pattern uses a Director class which is responsible for building a map containing the row objects parsed from the ".csv" files. To do this, the Director calls a Row Builder class, which will build a particular row object (either for a Teaching, Grants, Publications or Presentations ".csv" file) and return it. To build the row object, the Row Builder calls an Attribute Retriever class that will return specific attributes from a given row. For example, the Attribute Retriever may return a name, date, string, integer or boolean depending on the row field requested. These returned attributes are assigned to the member variables of a row object in the Row Builder class.

Inspections

The reconciled inspections can be seen below. Please refer to the “Inspections” folder for the individual inspections completed prior to reconciliation.

cs3307a – Object oriented analysis and design

Design Inspection Instrument (FINAL)

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. One is your analysis. The other is 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 of the system to be considered for inspection:

- With reference to Appendix B – Dashboard Screens, take Demo 1 feature, focusing on that part of the code that produces one Dashboard summary.
- Visualization code is out of scope of this inspection.

++++++++++++++++++

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: Classes were compared and seem to be consistent for the most part
Comment on your findings: Inconsistency in List Builder (it is missing for everything except Grants)

Functionality:

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

☐ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Each class does as intended
Comment on your findings: Each class serves a very specific function, ensuring each does as it's required

Cohesion:
Do the methods encapsulated in each programmed class, together perform a single, well defined, task of the class? (High-Cohesion: 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: Program has high cohesion
Comment on your findings: Classes do not take anything they do not need, code is efficient

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

☐ Yes ☐No ☐Partly (Can be reduced)

Comment on your analysis: Program does not have high coupling
Comment on your findings: Classes only use what they need, there is no excessive inter-dependency

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: Classes serve a specific purpose and has decently minimal connections with other concerns

Comment on your findings: Classes could have been broken down a bit more, but it is not necessary and would not enhance the code too much

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

☐ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Classes do not use private methods, code is functional without them
Comment on your findings: There could have been use of more separations, but it was not necessary

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: Classes that are not specific to the 4 file types are easily reusable
Comment on your findings: The classes associated with the four file types cannot be reused because they are optimized for each one. The other classes have great reusability (e.g. visual classes)

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

☐ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Almost all classes are easily identifiable and understandable
Comment on your findings: Only classes that seem complex are director and main window

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

☐ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Comments are well written for ease of understanding
Comment on your findings: All classes had comments explaining functions and complicated lines of code

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: This class would not be difficult to enhance
Comment on your findings: Some classes may need alterations in other classes to be enhanced

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: No inefficiencies in code
Comment on your findings: All loops are necessary and do not cause inefficiency in the code

Depth of inheritance:
Do the inheritance relationships between the ancestor/decendent 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: Lack of inheritance
Comment on your findings: Program did not need inheritance

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: No parent class has too many children classes
Comment on your findings: Inheritance would have complicated the code, was considered to be used for the rows but the file types did not have enough in common attribute-wise

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.: _____-
_____

(Note: expand here as necessary for each scenario)

Scenario #1: User selects and opens a file
Frequency: High
End-user trigger: From the main window, user selects the file from its directory
Expected outputs:

- If the csv file selected is correct, program updates and generates the plus-minus list for the file
- If the file is incorrect, program does nothing and waits for user to select another file

Key features:

- Main Window
- Director
- Builder
- Row Object
- Error Checker
- List Builder
- Plus Minus List

Comments: Fast response time, program functions as expected. The process is identical for each of the four file types, just goes through different classes to differentiate between the attributes

Scenario #2: Generating a bar graph
Frequency: Medium
End-user trigger:
- Provide first and last name
- Select date range
- Click on bar graph button

Expected outputs:
- A bar graph should be generated that compares the number of publications based on type and date
- If there is no data in the selected date range, a blank graph is generated
- If the name does not exist, the program gives an error

Key features:
- Main Window
- Bar Chart Dialog

Comments: Program handles errors very well, and functions as expected. The process is the same for generating a pie graph, only difference is clicking the “pie graph” button instead

Scenario #3: Printing a pie graph
Frequency: Low
End-user trigger:
- Provide first and last name
- Select date range
- Clicks on pie graph button
- Click print button

Expected outputs:
- Print dialog box is opened respective to system
- Any errors would be handled by the printing drivers

Key features:
- Main Window
- Pie Chart Dialog
- Main Window (QT)

Comments: Specific function operates well. Kept as simple as possible, errors are handled by the print drivers.

Development Plans

The development plans for this project are broken into two parts. The first is the Agent-Task summary which can be seen below. Note that the colour green is used to identify completed tasks, and that each agent's first C++ programming requirement is honored for the task coloured red.

		Assigned To:									
Task	Description	Colin Costello	Eric Lefebvre	James Crocker	Junwon Seo	Kevin Tawaststjerna	Lankesh Patel	Larsen Burchall	Peter Pfoertsch	Song Gao	Yiming Guan
System Design	Decisions on classes, interfaces between classes etc. This refers to the overall design of our program.										
Class Diagrams	Diagrams of classes to be used in the program. Related to the above task.										
Case Diagrams	Required for v1 and v2 submissions.										
Sequence Diagram	Required for v2 submission.										
Package Diagram	Required for v2 submission.										
Design Pattern Description	Required for v2 submission.										
C++ Implementation Justification	Required for v2 submission.										
File Selection/Location	Both the GUI and programming work that relates to prompting a user for the type of file, asking for the file name and locating the file.										
Row Objects	Objects to contain the necessary data to be pulled from rows in the excel file. One for each file type.										
Row Builder (Grant Only)	This builder is responsible for taking a given Grant row, and using the attribute builder to create a Grant row object.										
Attribute Builder	This builder will take a row and location, returning either a name, date, or primitive type like a string or int.										
Error Checking (Grant Only)	Determine if a particular row contained incorrect data.										
Director Class	This is the class that oversees map creation, row building, error checking etc. The rows of the file will be read here.										
Object for +/- List	This is a simplified object to be passed to the +/- List GUI for display.										
Object for Graphs	This is a simplified object to be passed to a Bar or Pie Graph for display.										
Builder for +/- List (Grant Only)	This builder is responsible for building the objects to be used for the +/- list. Only the Grant .csv file will be completed under this task.										
Builder for Graphs (Grant Only)	This builder is responsible fot building objects to be used for the graphs. Only the Grant .csv file will be used at this time.										
Map Construction (Grant Only)	This task pertains to building a map of all Grant row objects.										
Remaining Builder for +/- List	Same as previous builder for +/- list objects, except for the remaining three file types.										
Remaing Builder for Row Objects	Same as for previous builder for graph objects, except for the remaing three file types.										
Remaining Row Builders	Same as the row builder above, except implemented for the other three file types.										
Remaining Error Checking	Same as the error checking above, except implemented for the other three file types.										
Remaing Builders for Graphs	Same as previous builder for graphs, except expanded to include other file types.										
Remaing Map Construction	Same as the map construction above, except implemented for the other three file types.										
Main Window	Main window to display all elements of the program. (Including +/- List, graphs, date range filters, menu options etc.)										
"+/-" Display List/Tree	Displays the curated information from the .csv in an expandable +/- list format.										
Bar Graph	Displays data from the .csv as a function of time for a specific person.										
Pie Graph	Displays a comparision of attributes of a specific person over time.										
Graph Printing	Allows the user to print the data held in a graph.										
Proper File Commenting	Ensuring the code is properly documentted.										

Green - Completed
Blue - Planned
Red - Everyone's First C++ Requirement Achieved

The second part of the development plans is the timeline. Note that again, green is used to identify completed tasks.

		Date by Half Week (Sept - Dec 2015)																					
Type	Description	28-Sep	01-Oct	05-Oct	08-Oct	12-Oct	15-Oct	19-Oct	22-Oct	26-Oct	29-Oct	02-Nov	05-Nov	09-Nov	12-Nov	16-Nov	19-Nov	23-Nov	26-Nov	30-Nov	03-Dec	07-Dec	09-Dec
Non-Technical	Requirements Understood																						
	Development of System Design																						
	Allocation of Tasks																						
	Submission 3 Requirements Questions																						
	Submission of 2 Systems Questions																						
	Class Diagrams																						
	Case Diagrams																						
	Design v1 Accomplished																						
	Submission of Stage 1																						
	Design v2 Accomplished																						
	Submission of Stage 2																						
	Inspection and OO Metrics																						
	Final Report																						
	Design v3 Accomplished																						
	Submission of Stage 3																						
Back End Functionality	File Selection/Location																						
	File Reading																						
	Row Objects																						
	Row Builder (Grant Only)																						
	Attribute Builder																						
	Error Checking (Grant Only)																						
	Director Class																						
	Construction of Map (Grant Only)																						
	Objects for "+/-" List Display (a)																						
	Objects for Graph Display (b)																						
	Builder for (a) (Grant Only)																						
	Builder for (b) (Grant Only)																						
	Remaining Builders for (a)																						
	Remaining Builders for (b)																						
	Remaining Row Builders																						
	Remaining Error Checking (other file type)																						
	Remaining Map Construction (other file type)																						
Front End Functionality	Modification Based on Feedback from v1																						
	Modification Based on Feedback from v2																						
	File Selection/Location																						
	Main Window																						
	"+/-" Display List/Tree																						
	Bar Graph																						
	Pie Graph																						
	Graph Printing																						
	Date Range Selection																						
	Name Entry																						

Green - Completed
Blue - Planned

Lessons Learned

The following sections outline the lessons learned throughout the project.

1) The top three lessons learned are as follows:

    1) Software development using third-party libraries. Team members not being able to run our program made development and testing difficult and caused problems when integrating code. In retrospect, our team should have decided on a specific development environment and picked a common installation version of all third party libraries before we started developing. If we all downloaded and installed the same version of qt we would not have had this problem.

    2) Inheritance and polymorphism. We ended up with a fair number of files in our final product. Initially we had tried to use inheritance and polymorphism to reduce the number of classes and maximize the amount of code reuse, but in practice this proved to be difficult. Even though a lot of the tasks seemed similar which hinted at code re-use we found the algorithms for completing the different tasks unique enough to warrant their own classes.

    3) Organizing meeting times. The size of the group and the availability of group members made organizing a common meeting time difficult. In a project course like this one, it would have been great if there was a lab component where the whole class was scheduled for a lab and therefore all members would be guaranteed available during that time for a team meeting or group work.

2) Documenting the system design in the beginning was very helpful. Our team spent the first two meetings designing the system and discussing how the components worked together. After designing and documenting the system, all team members were on the same page and each team member had a high level understanding of how the system worked as a whole. Each team member understood the purpose and function of their assigned coding tasks.

Documenting the system first allowed our team to predict and identify the areas of increased complexity and increased programming difficulty. This enabled our team to allot the appropriate amount of time and resources to these difficult areas.

Once we identified our classes and their interfaces, our team was able to easily and evenly distribute the workload. Team members could code to an interface allowing members to work independently.

Taking the time at the beginning of the project to develop the design, identify classes and create interfaces, enabled our team to work as an informed cohesive unit. The time and effort spent designing and planning in the beginning, set our team up with the knowledge required to independently complete their assigned programming tasks.

3) Both liaisons agreed that customer interactions went smoothly. All incoming correspondence was received, reviewed, and promptly replied to. These interactions were courteous and professional.

4) Our team was well organized thanks to our project manager. Tasks were identified and the workload evenly distributed. Two group meetings a week increased the odds of team members being available to attend at least one of the weekly meetings. In the weekly meetings, task completeness was queried and any potential trouble spots were identified and appropriate resources were re-allocated. For the most part, team member performance was satisfactory. Anytime a team member couldn't complete an assigned task, other group members would jump in and pick up the slack. The only difficulty was finding meeting times that worked for all members. Aside from the first two meetings, we were unable to get all ten team members together for a meeting.

5) For the future generations of students taking the 3307 class, it is recommended that first and foremost the size of project teams is reduced to improve the experience. Coordinating schedules with so many people was extremely difficult, and since there were so many stakeholders in the project, there were more people to depend on. This increased the risk of the project not being completed on time, because now everyone's work was strongly dependent on their team member's work. As a result, small absences from team meetings due to assignments and midterms from other classes pushed back work for everyone.

Additionally, it is recommended that more feedback regarding the application being developed be provided throughout the course of the project. This group had interactions with the project evaluators halfway through the term, but this interaction was limited to describing how the team had been working up until that point. It was not until after the second submission deadline that the team received feedback on the application developed.

6) Although many group members began the project with an unfamiliarity with C++, and were able to practice using this language throughout the course of the term, the major learning for this project came from working in a group of such enormous size. Everything from coordinating schedules, dividing up the project and following tight deadlines proved difficult. Although the origin of many of these issues can be attributed to busy class schedules in addition to the 3307 project, still many of the issues stemmed from working with so many people.

As a result, this team learned how to break the project into manageable pieces which 1 or 2 people could handle independently, and subsequently combine with the other pieces of the project. Through this process, the team also inadvertently made the code more modular which had the added benefit of simplifying the debugging process and code maintenance. Since the project was broken into many pieces, the importance of deadlines became obvious. Early on in the project, it was believed that the team would be meeting frequently enough that we would not need to set frequent deadlines. However, this also caused many group members to take more time than they needed to complete trivial tasks. Fortunately, this realization happened early enough in the project that it could be reversed during the later half of the term.