cs3307a – Object oriented analysis and design

Design Inspection Instrument

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: _By following the class diagram and examining the code of the program, we can see
Comment on your findings: _The class diagram clearly captures the classes and interrelationships of the application

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

✓Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: _The classes all work as defined, no class has extenuating functions not related to the purpose of that class.
Comment on your findings: Checked the function declarations and function prototypes of the class and determined whether or not it was appropriate for the declared class.

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: __For the most part, the names of the class give a large idea as to what sort of functions will be found inside the class, as shown when we look at the code within the classes.
Comment on your findings: _Each class contains closely-related functions, or at least functions that perform a purpose related strongly to the class itself going by the class names. Publications data gets loaded in load_csv, verified in verify_csv, and analyzed in analyze_csv. We see that the data itself is stored as a dto in dto.cpp.

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: _Our program is slightly coupled but not in the conventional sense. The flow of the program is specified in a linear flow whereby the next classes are called once the previous classes are finished. This is more of a design metric as it is used to ensure the user is using the program as intended. For example, the data must first be loaded, and the classes required for data verification are called in order to allow the user to finish the data. Afterwards, the analyze page will generate the data structures for the visualization classes. In these cases the classes require on the execution of the previous classes but this is as intended.
Comment on your findings: Followed the execution through the program with the debug menu.

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: _It is obvious based on class names what the class is used for. Each class covers specific functions designed to accomplish a specific task. Thus each concern will be covered by a single class, or a couple of classes if the concern is broad enough.
Comment on your findings: _The various concerns are split up such that all similar concerns are contained together within a single class. Functions are split up between general functions used in multiple concerns, and those more specific which may reference these general functions. Publications data has its code made into graphs and trees in analyze_csv through the use of different classes, like bargraphadapter.cpp, which is designed to specifically handle the concern of creating a bargraph.

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

✓Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: _Yes most classes have proper accessors and getters.
Comment on your findings: Checked the function header declarations.

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: This program is very specific. Most classes are designed for the analysis and verification of csvs. However, the data structures used to store the data for visualization and the functions used to generate the error lists are all modular and reusable.
Comment on your findings: Checked the usage of each class. Some classes are used by multiple others.

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

✓Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: _Since the classes have high cohesion and are named appropriately, as are the methods within the classes, it's easy to identify and understand what a class and what each method within it does. There are many comments that also clarify what a function or variable is for.
Comment on your findings: _Each class and its methods have relevant names. It's easy to follow the path the data will take by examining the functions, starting with load_csv, so the code is definitely simple to examine. We can see once we begin examining analyze_csv how the publications data will have its data become a bar graph, a tree, etc.

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

☐ Yes ☐No ✓Partly, can be improved

Comment on your analysis: _The header files are heavily commented. However, the actual cpp files could use more comments within them at the time of writing up this inspection.
Comment on your findings: _Although the header files explain each method and variables, at least when they're not obvious, some of the cpp files could use more comments to explain the specifics of the functions. This is obvious comparing files like csvfieldvalidator.cpp, which contains a lot of comments, to csvlinevalidator.cpp, which contains no comments.

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 code is well written to accommodate changes and maintenance in the future. It has lots of general usage templates for incorporation of different data types later on.
Comment on your findings: Checked the amount of Templates and void pointers in the code.

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: _Looking through the cpp files we can see that the code is very efficient. It uses general types and cases to cut down on how much code is required. Methods are typically fairly short in length.
Comment on your findings: _There are very few nested loops located within the code. Each method is straightforward in what it does and most are short in length. There are quite a few methods, but since the code uses a general csv data type and cases in the larger methods when necessary, it cuts down on the amount of code required for different data types other than just the publication data.

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: No, the linear flow is very efficient for data processing. The nested loops used in the data structure generation can be parallelized.
Comment on your findings: No inefficiencies found when loading/running analysis. All csvs load in less than 1 second.

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: _Most classes have no children classes (or parent classes). However, the csvfieldvalidator.h file has multiple children classes, each designed for a different purpose.
Comment on your findings: _Since the csvfieldvalidator.h file has multiple classes with it, we could probably improve upon this to decrease the number of these subclasses.

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: Load a Publication CSV File
Frequency: High
Starting Point: At initial start-up screen for application
Expected Output: We will move to the Verify window, which will display all errors.
Walkthrough:

- User clicks to continue, brings up load screen from load_csv.cpp
- User clicks publications button, bringing up window to select a publications csv file. Relevant method in load_csv.cpp performs this operation.
- User selects the relevant file.
- Program moves to the Verify window by calling for verify_csv.cpp.
- All errors are displayed by program

Title: Ignore All Errors in CSV File
Frequency: High
Starting Point: At Verify screen for a CSV
Expected Output: Analyze window displays a collapsible tree of information, as well as relevant graphs.
Walkthrough:

- User clicks Ignore All to skip any errors in the CSV.
- Program moves to the Analyze window by calling analyze_csv.cpp
- Inside analyze_csv.cpp, methods are accessed to create and display the graphs and trees.
- Tree can be collapsed or expanded, while different graphs can also be selected.

Title: Load Second, Different CSV Type, Ignores all Errors
Frequency: High
Starting Point: At Load screen after another CSV has already been loaded and verified
Expected Output: Analyze window displays tree and graphs for new CSV type, but can swap tabs between new CSV type information, and the old CSV's information.
Walkthrough:

- User clicks Presentations button on load screen in load_csv, after a Publications CSV has already been loaded, verified, and analyzed. This brings up a window to select a presentation csv file. Relevant method in load_csv.cpp performs this operation.
- User selects the relevant file.
- Program moves to verify window.
- User hits Ignore All. Program moves to Analyze window.
- Both Publications and Presentations tabs can be selected. The data persists, allowing user to compare the two by switching between these tabs.
- Publications and Presentations tabs display relevant trees and graphs.

Title: Load Incorrect CSV Type
Frequency: High
Starting Point: Load window
Expected Output: Error message indicating that user attempted to load the wrong type of CSV.
Walkthrough:

- User clicks Publications button on load screen. This brings up a window to select a publication csv file. Relevant method in load_csv.cpp performs this operation.
- User selects a csv of presentation data instead.
- Error message is displayed, indicating the file does not have the correct headers. User is unable to move to Verify window.

Title: User Decides not to Verify Data, Moves Back to Load
Frequency: Low
Starting Point: At Verify screen for a CSV
Expected Output: User prompt, then load screen.
Walkthrough:

- User clicks Load to return to the Load window.
- Program displays prompt alerting user that the current csv's data will be lost.
- Function in verify_csv is called to return the user returns to the Load window (load_csv.cpp).

Title: User Corrects Some Errors, Ignores the Rest
Frequency: Normal
Starting Point: At Verify screen for a CSV
Expected Output: Analyze window displays a collapsible tree of information, as well as relevant graphs. Corrected data is also included in this.
Walkthrough:

- User enters information in relevant fields that contain errors for a few of the data entries displayed in Verify.
- User hits Confirm Changes. verify_csv.cpp uses function to re-check the changed lines for errors. All properly-corrected errors are thus stored, while incorrect lines remain in the list.
- User hits ignore all. Remaining errors are thus ignored.
- Program moves to Analyze window. User clicks to expand tree to locate and confirm that their