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: _ by comparing the code within the classes to the project requirements, it is easy to determine what operations can be performed_____
Comment on your findings: __each class is used for one specific purpose, whether its loading data, verifying data, obtaining graphs and information from the data, etc. The publications data is handled largely as a generic CSV data type, or else through cases when necessary._____

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 clases____
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: __moving through the code, we can see that few classes require information from other classes – or at the very least, they are not excessively linked together more than would be necessary___
Comment on your findings: __in general, classes perform functions related solely to the class – although there are some functions that require other classes, in general the coupling is quite low. Verifying the publications data in verify_csv requires that we send the data into this class, then use the field validator and line validator classes to validate each field and line in the csv._____

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: _examining header files reveals private and public methods, as well as any private or public variables_____
Comment on your findings: __there are no public variables. Instead, get functions are used to obtain the relevant private variables. Most methods are public for use by other classes when necessary, for example bargraphdata contains only public methods so that the analyze.cpp function can access it to display the bargraph in the analyze window.___

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: ___some classes such as load_csv could be used in a variety of applications, due to its general function of loading a csv. Since the classes are designed to employ very specific functions, this makes it easy to reuse them for the same or a similar task in another application.__
Comment on your findings: __even within this project, we can reuse the same code for multiple csv types – not just publication, but also presentations, grants, and teaching. The high cohesion of the classes means we can easily reuse them in other projects.___

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: ____without knowing what sort of enhancements might be needed, it's hard to judge how easy it would be to enhance the code. However, most likely it would be, yes, due to the fact that much of the code uses cases or general types, and that there is very high cohesion within the code that would make changing or updating it quite simple._____
Comment on your findings: __depending on what sort of enhancements or updates would be wanted, it may be easy to implement them, or it may be hard. The code's high cohesion would make it easy to

perform enhancements on specific parts of the application. For example, we could make it possible to print a graph by right clicking on the graph, and the code we use to do this for one type of graph, like a graph of the publications, would allow it to work on all other graphs as well._____

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: ___looking through the code, it seems that all subclasses are only one level deeper than their parent class.__
Comment on your findings: ____If we look at the csvfieldvalidator.h file, we can see a that there are multiple subclasses within this class – however, they are all only one level deeper than their parent class. This is the most complex depth within the code._____

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 error-corrections are now included in the Analyze page.