cs3307a – Object oriented analysis and design

Design Inspection Instrument (List Builder)
Lankesh Patel

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.
- Visualisation 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: Compared classes in code to classes listed in class diagram
Comment on your findings: Classes are consistent between program and class diagram

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

[✓ ☐ Yes] ☐No ☐Partly (Can be improved)

Comment on your analysis: Check to see if class creates the lists correctly for the summary
Comment on your findings: Program correctly creates the necessary lists for the data summary

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: Check to see if the methods in List Builder are used together to perform a single, well defined task
Comment on your findings: The program has high-cohesion; every method is necessary and sufficient to create the list summaries

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: Check for shared variables and reliance/execution of other classes
Comment on your findings: The program does have high coupling, but it is not excessive.

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: Check program to see if it is a class with well-defined interface and cohesive functions
Comment on your findings: ListBuilder class serves a specific purpose and has minimal connections with other concerns

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

[✓ ☐ Yes] ☐No ☐Partly (Can be improved)

Comment on your analysis: Check the access modifiers of methods within code
Comment on your findings: Methods in program is public, which is appropriate because they will be called by other methods

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: Check generality of code to see if it could be used outside of this program
Comment on your findings: Has very similar functionality as other classes in program, but differs based on the type of .csv file requested

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

[✓ ☐ Yes] ☐No ☐Partly (Can be improved)

Comment on your analysis: Check structure of code, comments, spacing
Comment on your findings: Functionality of class is easily identifiable and understandable

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

[✓ ☐ Yes] ☐No ☐Partly (Can be improved)

Comment on your analysis: Check for comments explaining the complex parts of the program
Comment on your findings: Class is easy to understand and follow; comments are done well

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: Check to see if attributes can be added/removed, if code is easily modifiable for updates
Comment on your findings: This class would be not be difficult to enhance

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: Check for poorly written code, see if anything could be simplified
Comment on your findings: No inefficiencies, code runs smoothly

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: Check all relationships this class has with other classes, measure depth of hierarchy
Comment on your findings: Class' depth of inheritance is appropriate, not 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: Check all children classes of inspected class
Comment on your findings: Does not have too many children classes

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.

Scenario #1
Title: Printing list
Anticipated Frequency: Medium
Starting Point: ListBuilder::printList method is called
Expected Output: Method prints entire list
Key Features of System:

- List is iterated through and stored
- Loops through iteration to print name and parameters

Comments: Given a functioning list, method works correctly

Scenario #2

Title: Adding member to list
Anticipated Frequency: High
Starting Point: Program attempts to add next member to list object
Expected Output: Member is successfully added to list
Key Features of System:

- Method uses variables associated with new member to determine its exact type
- Attempts to add member to list with all of the necessary attributes
- Completes and updates new list

Comments: Member is correctly added to list

Scenario #3
Title: Building a list
Anticipated Frequency: High
Starting Point: Program attempts to build a list object for Grants & Clinical Funding
Expected Output: List is built correctly with all necessary attributes
Key Features of System:

- Creates main list head
- Creates major list heads
- Creates minor list heads
- Populates variables with data from .csv file

Comments: List is built correctly, given the right .csv file (Grants & Clinical Funding)

END.