cs3307a – Object oriented analysis and design

Design Inspection Instrument

Builders

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: Class Diagram shows good setting
Comment on your findings: simple, so as expected

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: every Builder builds run like intended
Comment on your findings: each is specific and simple

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: Only need row and col index
Comment on your findings: rest is done in builder

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: Need supporting and Director
Comment on your findings: but this is necessary

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: Each builders focuses on its concern
Comment on your findings: could maybe simpler

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

☐ Yes ☒No ☐Partly (Can be improved)

Comment on your analysis: No public or private constructs
Comment on your findings: 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: For specific files
Comment on your findings: could have been made better

Simplicity:

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

☑ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Each builder does what it says
Comment on your findings: simple class, no surprise

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: plenty of comments, even in code
Comment on your findings: easy to understand functions

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: simple to add new conditions
Comment on your findings: requires good understanding of logic

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 loops
Comment on your findings: not needed

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 inheritance
Comment on your findings: not needed

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 children needed
Comment on your findings:

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)

END.

cs3307a – Object oriented analysis and design

Design Inspection Instrument

Row Objects

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: Used as an diagram
Comment on your findings: very easy [illegible]

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 row does as required
Comment on your findings: row objects are easily instantiated

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: obsolete with raw builders
Comment on your findings: simple to understand

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: Raw object is used
Comment on your findings: might use robot object in raw builder

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: simple use
Comment on your findings: no

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

☐ Yes ☒No ☐Partly (Can be improved)

Comment on your analysis: None are
Comment on your findings: not used / 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: All raw objects are specific
Comment on your findings: can't use them again unless you really make similar raw objects

Simplicity:

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: simple, as done
Comment on your findings:

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: comment are used often and easy to understand
Comment on your findings: good way to use

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: Row objects can be improved
Comment on your findings: but hard to do so, need to be similar

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 loops
Comment on your findings: not needed

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: None
Comment on your findings:

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: none used
Comment on your findings: _____

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)

END.

cs3307a – Object oriented analysis and design

Design Inspection Instrument

Supporting

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: low relationships, easy to spot
Comment on your findings: no surprise

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 does as title intends
Comment on your findings: no surprise here

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: Supporting are used by [illegible] frequent,
Comment on your findings: needed may be too much

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: Use is as intended
Comment on your findings: more or less would be bad

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: each class has specific use
Comment on your findings: very clean

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

☐ Yes ☒No ☐Partly (Can be improved)

Comment on your analysis: None used
Comment on your findings: not needed

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: Error checker and attribute setter could be reused but not.
Comment on your findings: no surprise, and very good job. Coupling's unreusability is unavoidable

Simplicity:

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Specific short classes
Comment on your findings: very contained

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: comments on class & function
Comment on your findings: explains all in all

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: simple, can be done
Comment on your findings:

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 loops
Comment on your findings: none needed

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 inheritance
Comment on your findings: not needed

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 children classes
Comment on your findings: none needed

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)

END.

cs3307a – Object oriented analysis and design

Design Inspection Instrument

Visual - Only +/- list

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: All classes in Diagram are used by Main window
Comment on your findings: Good correspondence, except with extra list builds,

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: Used the +/- list for each type
Comment on your findings: All worked and wouldn't without

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: Each Class has specific function to fulfill
Comment on your findings: Main window has most functions, but all related to window

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: Only MainWindow calls from other classes
Comment on your findings: Mainwindow only takes what it needs

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: Each class has only function to [illegible]
Comment on your findings: Class Design is made with concerns in mind

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

☐ Yes ☐No ☒Partly (Can be improved)

Comment on your analysis: None found
Comment on your findings: Not needed, but few objects probably could have 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: None specific file-type classes can be used again
Comment on your findings: the file type specific could be reused but with great work

Simplicity:

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

☒ Yes ☐No ☐Partly (Can be improved)

Comment on your analysis: All functions commented on function
Comment on your findings: Easy to see what each function does

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

☐ Yes ☐No ☒Partly (Can be improved)

Comment on your analysis: All functions commented
Comment on your findings: Could use more comments inside functions to better understand how function works.

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: Functions are self-contained
Comment on your findings: more functions and extra types can be made

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: Few loops, fewer nested loops
Comment on your findings: Most loops only go once through info

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: Inheritance is only 1 level deep
Comment on your findings: This part is pretty closed

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 Children
Comment on your findings: easy to see

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)

END.