Daniel Brian Wiegold Stannard

17736382

Object Oriented Programming

ASSIGNMENT 2

Contents

[Introduction 2](#_Toc127203641)

[Use Cases Analysis 2](#_Toc127203642)

[UML Class Diagram Analysis 8](#_Toc127203643)

[UML State Diagrams Analysis 10](#_Toc127203644)

[Conclusion 16](#_Toc127203645)

[References 17](#_Toc127203646)

[Table of Figures 17](#_Toc127203647)

# Introduction

This task requires the analysis, critique and comparison of a set of diagrams and listings designed to satisfy a specific scenario, the set given was identified as number 11 and will be referred to as ‘Donor’. These will be analysed and compared with those previously produced for the exact same scenario, these diagrams will be referred to as ‘Host’ purely for the sake of identification in any comparisons made.

# Use Cases Analysis

The analysis of the Use Cases listing will mostly be a study and comparison of whether all requirements of the application to be developed are met, this will look at the level of detail used to meet all requirements along with the document suitability also.

Donor Use Cases listing: (Figure 1, Figure 2)

When opened it quickly became apparent that there was no ‘Requirements’ statement explaining what the Use Cases listing was trying to fulfil, this would normally be essential to the reader because one couldn’t assess whether the listing is true to the scenario or not.

Another thing noticed with this diagram is not really a fault but more of an observation, the order of the listing follows the order of the scenario specification. While this is fine, this could possibly be improved upon, perhaps study the scenario in detail before writing the Use Cases listing. This could maybe take the form as more of a walkthrough, where the writer and reader of the listing could visualise using the system from the start. For instance, start at the logging in and what one would expect to see there, once logged-in what options would one expect to see available to them. This approach possibly could possibly reduce the chance of overlooking anything, it would probably be easier for the reader to follow as on a large project the developer may only have this detail to follow and not that of the original spec.

That said, the Donor listing appears to cover everything listed in the specification without anything major being missed out, the level of detail is mostly good with only some minor points spotted. These are:

1. This listing shows no consideration for the creation of a user account, neither is this mentioned in the original specification. However, this would be an essential part of the system, logging-in is mentioned but nothing about the creation of a user account to log in to. Perhaps an initial default ‘admin’ (typically admin, admin) account can be expected, but not a ‘user’ account.
2. In section 4. there is no mention of ‘game developer’ information being displayed in the list of entries.
3. In section 5 Part 3 it mentions ‘shown a list of stored passwords that match the search term’, the search term would be the entry (game/website/application) name, this would probably be a unique key field to that user so only one should exist.
4. In section 7. Although there is a mention of the user selects ‘generate random password’ as required in the specification, it is not clear at what point this is done or to what it’s applied.
5. The term ‘User’ is used throughout this listing, a suggestion for section 9 of the listing maybe to use ‘adminuser’ or similar to clarify and differentiate between the two user types.

The points mentioned above may appear to be minor or even petty, also they may be of little significance if the same person(s) are responsible for every step of the development from start to finish. However, in the case of passing on the design to the next team, or even not seeing it through for whatever reason, these minor issues could be major to the final outcome.

Text

Description automatically generated

Figure - UML Use Cases Donor Part 1

Graphical user interface, text, application

Description automatically generated

Figure - UML Use Cases Donor Part 2

Host Use Cases listing comparison: (Figure 3, Figure 4, Figure 5)

Point 1: in the critique of the Donor listing where a user account is assumed, this is clearly identified and addressed in section 1 of the Host listing, it appears to be well covered using a detailed multi-level approach.

Point 2: again, in the Donor listing where there was no provision for ‘game developer’ information, this is identified at point 2:1.3.3 in the Host listing.

Point 3: the Donor listing mentions a ‘list’ of results matching the search criteria being displayed, the Host listing mentions clearly the process up until the search is made 5:1.1.2, but lacks any detail regarding the displaying of the results or any ‘not found’ error situation.

Point 4: with the Donor listing there is no clear listing of where the randomly generated password fits into the process, the Host listing clearly identifies this in an appropriate point of adding a new entry 2:2.

Point 5: the Donor listing does not clearly identify that the ‘user’ referred ton in section 9 is actually ‘admin’, this is made quite clear in section 7 of the Host listing.

Text

Description automatically generated

Figure - UML Use Cases Host Part 1

Table

Description automatically generated

Figure UML Use Cases Host Part 2

Table

Description automatically generated

Figure - UML Use Cases Host Part 3

Further analysis of the Host Use Cases listing.

Once a new entry is chosen and the appropriate detail entered for the type of entry, the final step would be to ‘save’ the entry, the Host listing offers separate ‘save’ options for each type of entry 2:3\*\*. This is slightly confusing and could be possibly simplified with a single ‘save’ option.

Conclusion:

It appears that the Donor listing covers most aspects quite well but perhaps lacks some minor detail at some points.

The Host listing appears to consider the bigger picture overall (user account creation) , and the multi-level approach makes it easier to identify exactly where in a process a particular action needs to happen, however, it does have some faults.(Cockburn, 2000)

# UML Class Diagram Analysis

Donor UML Class Diagram: (Figure 6 - UML Class Diagram Donor)

The initial analysis of the Donor UML Class diagram was difficult, this is because there was one type of arrow used to join all connections between classes, this made it difficult to identify the relationship or the dependency between classes joined by these arrows. There are no clear signs of any meaningful inheritance and the DRY (Don’t Repeat Yourself) was not used. This is clear to see with the three different classes WebsiteLogin, AppLogin and GameLogin because they share so many common attributes and methods which were repeated several times. All attributes in all classes are private ‘-‘ with no method to get or set these attributes, In general this would make it impossible to read or update the values of theses attributes in the main body of the application or other objects. Because this diagram had no clear structure, it was difficult to identify any efforts to adopt SOLID principles. There has been an effort made to show the attribute types, along with those passed to and from methods. (Columbus et al., [no date])

Recommendations on where to start improving this diagram would be:

1. Add ‘setter and getter’ methods for the private attributes to make them accessible to other classes or the main body of the application.
2. Consider using ‘inheritance’ to avoid replication of attributes and methods (DRY).
3. Use appropriate relationship arrows to imply inheritance, dependency, association , multiplicity etc.. to the reader/developer.

Diagram

Description automatically generated

Figure - UML Class Diagram Donor

Host UML Class Diagram: (Figure 7 - UML Class Diagram Host)

Point 1: The Host diagram show appropriate setters and getters for all private attributes.

Point 2: Inheritance correctly and clearly indicated in the Host diagram.

Point 3: The Host diagram uses appropriate relationship arrows and links.

Flaws: Further analysis of the Host diagram shows inconsistent listing of attributes. In classes , Website\_record, Game\_record and User the types are listed before the attribute name while other classes are listed correctly. Although this is incorrect, if it was the only diagram available without correction, one would still be able to use it as all the information is present. There is to also another inconsistency within this diagram, this is where the ‘type’ attribute is declared in the Application\_record’ ,‘Website\_record’ and ‘Game\_record’ classes are declared with different visibilitiy indicators, it is believed all should be ‘protected #’. The fact that the ‘Data\_Store’ class is also host to the ‘decryptPassword()’ method would imply that SOLID principles have not been strictly adhered to.

This host diagram clearly shows the ‘Application\_record’ superclass being inherited by the ‘Website\_record’ and ‘Game\_record’ sub classes, another example of inheritance is where ‘Admin’ subclass inherits the ‘User’ superclass.(E. V. Sunitha, [no date])

Diagram

Description automatically generated

Figure - UML Class Diagram Host

Conclusion:

The analysis of both UML Class Diagrams show:

The donor diagram has very little in the way of usefulness with the absence of many aspects and incorrect relationship arrows so has no real structure to follow.

The Host Diagram has many positive aspects to it, but also has several typo errors and inconsistencies, along with non-ideal design aspects (no SOLID). (Snyder, 1986)

# UML State Diagrams Analysis

Donor state diagram1 (adding new password). (Figure 8 )

This diagram is a little unclear, as from the ‘Start’ box there is a self-transition loop where if the ‘Add new password’ option is selected then it would loop around back to the ‘Start’ state. It is difficult to see under what user input would the system move beyond the ‘Start’ box. Again, at the ‘Enter Details’ state if ‘Cancel’ is selected one would expect to ‘Return to Main Menu’ but the diagram shows looping back to same state, also following a successful ‘Save to Database’ one would expect to return to the ‘Start’ or and ‘End State’ symbol, neither were present.

Diagram

Description automatically generated

Figure - UML State Diagram for Adding New Password Donor

In comparison the Host state diagram(Figure 9) for the same process is much more detailed but probably clearer to follow, however, there is a possible typo error where the loop above the ‘enter record name’ state should probably read ‘Text empty’ so keep looping until a record name is entered. This diagram shows greater detail of a more complete process.

Diagram

Description automatically generated

Figure - UML State Diagram Add New Record Host

Again here in the Donor diagram (Figure 10 ) the functional intentions are unclear, the self-transition looping on the ‘Start’ state seems to loop back to itself once the ‘Delete password Option selected’, again its difficult to see how the system would progress beyond this point because it’s difficult to see how ‘Display list of Stored passwords’ is an event that would trigger a transition to the ‘Select password’ state. A similar issue is present at the ‘Select password’ state, where once a password is selected it would loop back to the same state. There seems to be similar issues around the ‘Confirm Deletion’ state. If this process was to work then one would expect the ‘Delete password from database’ to be the final event, so the ‘Database’ state has no return path to the start or a end state symbol.

Diagram

Description automatically generated

Figure - UML State Diagram for Deleting Password Donor

The Host diagram (Figure 11) for the similar function is more detailed but fairly clear to follow, states and transitions have sensible labels and the flow easy to follow. This diagram is a little more involved than that of the Donor diagram, this diagram implies that the data store in a form of a list or table. It appears that the list is stepped though one record at a time, comparing the search criteria with each row of the list. This diagram shows a similar error to the previous host diagram, the self-transition around the ‘Enter name’ state should not include the word ‘not’, this is probably a typo and not a logic error, because if a name was entered instead of transitioning to the next state it would loop back on itself.

Diagram

Description automatically generated

Figure - UML State Diagram Search for and Delete Record Host

This Donor diagram for viewing password (Figure 12) seems to follow a similar pattern to previous diagrams analysed from the same source (the Donor), again, the ‘Start’ state has a self-transition which appears to loop back on itself when an event one would expect to happen occurs. This result of this will be, when the user selects the ‘View Password’ option from the UI, this will loop back to the ‘Start’ state instead of transitioning onto the what is probably intended to be the ‘View password state’ and beyond. So, it’s difficult to see how again this system will leave the start state.

Although there is no mention of ‘decrypting’ the password before display, there is however a transition labelled ‘View full Password details which is probably intended to cover this which is implied in Donor’s Use Cases listing. (*UML 2.0 in a Nutshell - Dan Pilone, Neil Pitman - Google Books*, [no date])

It appears that there is some confusion over the ‘transition’ text and the ‘state’ text.

Diagram

Description automatically generated

Figure - UML State Diagram for Viewing Password Donor

Although clearly labelled and easier to follow the flow, the Host diagram (Figure 13) for the function similar to that in Donor diagram (Figure 12) is a little confusing also. The diagram implies that data is stored in possibly a CSV type file, this is implied by the ‘Separate all field in records’ state, however the section after the decision diamond implies that only admin can view the (decrypted) password. This differs from the original scenario specification and Host Use Cases diagram, where the user can view the passwords too.

Diagram

Description automatically generated

Figure - UML State Diagram View Entries Host

Here is the analysis the Donor’s UML State diagram (Figure 14) for ‘Search for passwords’:

Here we again have some confusion surrounding the ‘Start’ state, one can imagine the user being presented with a UI which has an option to ‘Search for passwords’. The diagram implies that if this option is selected by the user that the function flow will simply return to the ‘Start’ state, it is probably safe to assume that the transition labelled ‘Display form to Enter search criteria’ is the intended path on selection of this option. Again, there appears to be a similar issue surrounding the ‘Enter search criteria’ state, where if ‘Search’ is selected the self-transition will simply return to the same state. If one can assume the ‘View’ state is the end of the intended function, there is neither an ‘end state’ or a transition bac to the start.

Diagram

Description automatically generated

Figure - UML State Diagram for Searching for Passwords Donor

This is the Host UML state diagram (Figure 15) for the same ‘Search for password’:

This diagram has plenty of detail with clearly labelled states and transitions, the flow is easy to follow but does contain the same error previously seen in similar diagrams from the Host. This is the ‘self-transition’ at the ‘Enter name’ state, this probably shouldn’t contain the word ‘not’, because if the ‘name’ is not empty, then it is probably a name intended for the search. In this case two transitions could be met at the same time which could cause errors or confusion.

Diagram

Description automatically generated

Figure - UML State Diagram Search for Password Host

# Conclusion

Following the thorough study, analysis and comparison of UML Class, UML State diagrams along with Use Cases listings from two separate sources, the importance of the accuracy and appropriate level of detail became very clear.

Starting with the Use Cases listing and how close it matches the original required specification of the application to be developed, this is crucial, if that is not true to the specification at that point then all steps from then on will not be true either.

The UML Class diagram has more effect on the performance, reliability and durability of the application, if it’s designed well and SOLID principles are followed then future expansion can be carried-out without disruption to the overall system.

The UML State diagrams are essential to get the correct functionality for each feature of the finished application, ‘Single Responsibility Principle’ of the SOLID principles can be used here to keep things simple and reliable for the developer.

To conclude:

Both sets of diagrams carried some useful information, but both had errors or lacked detail at some point. In general the Host diagrams and listing were the better overall.

Donor and Host Use Cases listings were reasonably comprehensive with maybe the Host containing slightly more detail, but perhaps with a little redundancy.

UML Class diagrams: both Host and Donor diagrams had errors, the Donor diagrams were difficult to follow because of the non-standard lines used to link all classes, there were other error also. The host diagram was better but still contained errors, there was some evidence that ‘Single Responsibility Principle’ of the SOLID principles were not rigidly enforced.

UML State diagrams: the Donor diagrams had many errors, these were mostly the result of confusing the state and transition behaviour, The host diagrams were far better and contained clear detail with some minor errors identified.(Arora, 2016)

Final conclusion:

The Host diagrams will be used for the development of the application.

# References

Arora, G.K. (2016) ‘SOLID Principles Succinctly Foreword by Daniel Jebaraj’. Available at: www.syncfusion.com. (Accessed: 16 January 2023).

Cockburn, A. (2000) ‘Writing Effective Use Cases’. 3

Columbus, B.• et al. ([no date]) ‘The Pragmatic Programmer your journey to mastery 20 th Anniversary Edition’.

E. V. Sunitha, P.S. ([no date]) *Automatic Code Generation From UML State Chart Diagrams*. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8600324 (Accessed: 13 February 2023).

Snyder, A. (1986) ‘Encapsulation and Inheritance in Object-Orlented Programming Languages’.

*UML 2.0 in a Nutshell - Dan Pilone, Neil Pitman - Google Books* ([no date]). Available at: https://books.google.co.uk/books?hl=en&lr=&id=Xg8wLPmt5CMC&oi=fnd&pg=PR7&dq=class+diagrams+uml+2.0++programming&ots=EyS93X9i5I&sig=ZfwrHKjjoUrl1Jm995vv5yq3LIc&redir\_esc=y#v=onepage&q=class%20diagrams%20uml%202.0%20%20programming&f=false (Accessed: 13 February 2023).

# Table of Figures

[Figure 1 - UML Use Cases Donor Part 1 3](#_Toc127203626)

[Figure 2 - UML Use Cases Donor Part 2 4](#_Toc127203627)

[Figure 3 - UML Use Cases Host Part 1 5](#_Toc127203628)

[Figure 4 UML Use Cases Host Part 2 6](#_Toc127203629)

[Figure 5 - UML Use Cases Host Part 3 7](#_Toc127203630)

[Figure 6 - UML Class Diagram Donor 9](#_Toc127203631)

[Figure 7 - UML Class Diagram Host 10](#_Toc127203632)

[Figure 8 - UML State Diagram for Adding New Password Donor 11](#_Toc127203633)

[Figure 9 - UML State Diagram Add New Record Host 11](#_Toc127203634)

[Figure 10 - UML State Diagram for Deleting Password Donor 12](#_Toc127203635)

[Figure 11 - UML State Diagram Search for and Delete Record Host 13](#_Toc127203636)

[Figure 12 - UML State Diagram for Viewing Password Donor 14](#_Toc127203637)

[Figure 13 - UML State Diagram View Entries Host 14](#_Toc127203638)

[Figure 14 - UML State Diagram for Searching for Passwords Donor 15](#_Toc127203639)

[Figure 15 - UML State Diagram Search for Password Host 16](#_Toc127203640)