CIS573 –Software Engineering

3rd Homework

Yayang Tian 24963298

Ning Xu 20139217

"UserInterface" Class

Main() Method

1. Code Smells & Solutions

(1)**Line 18-32 Long Method**: don't rely on other methods, and should be split out Unclear variable: Extract Method: **Solution:** we create a new method called showChoices(), which shows all the choices available.

(2)**Line 37-46, 49-55, 58-77 Long Method**: in the "switch-case" conditional sentences are too long, this are code smells of long method. We'd better summarize them as three methods.
Solution: Extract Method: we create three new methods, namely printByYear(), printByTeam() and printByRange(), which take "year", "team name", and "starting year and ending year" as inputs relatively, and print strings showing the results, in terms of year, team and range information respectively. Note that since "break;" cannot be used outside a loop, we have to replace it for "return" in printByRange().

(3)**Line 52 Naming**: the Variable name "which" is bad, it is not clear what it does.
Solution: Let's change the name "which" to "winOrLose" to indicate whether the user what to see the wining or the losing results.

(4)**Line 59-66 and 67-74 Duplicate code:** are apparently duplicate code because the only difference are "start" and "end". We should extract them as a method.
Solution: Extract Method: we create a new methods, namely getInputYear(), which take a string variable indicating whether the user what to set start year or end year, with no return value. The method firstly print out guidance to user, telling them to input starting/ending year, then set the variable startYear/ endYear with the input the user gives.

(5)**Line 90 and Line 96 Long Method**: The conditions are a bit too long, these are code smells of long method.
Solution: Decompose Condition: To make it a bit easier to read, we create a new method notEqualToQ(), return a Boolean. Return true if choice not equal to Q.

(6)**Line 80 and Line 81** : After all the modifications above, we find that case4 in the "swith-case conditions" are not consistent with previous cases, although it's not long at all.
Solution: Extract Method: To make it a bit easier to read, we create a new method printAllSeries, to show all the teams and the years they won World Series.

2. Removed Changes:

Since we can simply decrease the value of metric "lines of codes" by eliminating the Space line, we tried this first to make the improvement. However, in fact, eliminating the Space line will decrease the readability of codes, thus making codes hard to understand. Therefore, we back this change out and still use Space line to distinct each part of codes.

3. Prest Metrics:

We first compare the Prest Metrics between the original "UserInterface" Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	0.02	247	210	227	78	20	25	437	2890.6
Refactored	0.01	248	212	207	102	21	20	419	2908.9

Since we mainly refactor the method "Main()", we then compare the Prest Metrics between the original method and the refactored method.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	6.29	88	53	45	40	12	7.69	98	558.6
Refactored	5	45	25	25	18	10	7.14	50	240.3

4. Improved Prest Metrics

Clearly, in the method comparison, the metrics of the refactored method Main() are almost better than the original method.

(1)Length of code: since we extract many uncorrelated codes from the original method, the length of the refactored is improved largely. This has solved the Long Method problem;

(2)Operators & Operands: both operators and operands (except the unique operator count) has been reduced, which would make the refactored method is easier to read and understand.

(3)Halstead metrics: clearly, through refactoring, the Halstead metrics is improved a lot, which also indicates the refactored method is better than the original method in readability.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of the refactored class have gone a little worse.

This is due to we introduce many new small method into the refactored class while the original method only have 3 methods. Therefore the total complexity seems to be increased. However, since the functionality of all the small method is clear enough, the readability is not deteriorated in fact. Therefore, it may not be appropriate to evaluate the refactoring based on the whole class in our case.

showDataForYear() Method

1. Code Smells & Solutions

(1)**Line 102: Naming**: "ds" is not clear what this variable represents.
Solution: Lets' change "ds" to "allData" to indicate all the data from DATAFILE.

(2)**Line 105,106**: **Naming**: "list" is not clear what this variable represents.
Solution: Lets' change "list" to "allIncidenceList" to indicate all the Incidences created from from "allData"

(3)**Line 106,109: Naming**: "wsi" is not clear what this variable represents.
Solution: Lets' change "wsi" to "oneIncidence" to indicate one of the incidences in "allIncidenceList"

2. Removed Changes:

Since we can simply decrease the value of metric "lines of codes" by eliminating the Space line, we tried this first to make the improvement. However, in fact, eliminating the Space line will decrease the readability of codes, thus making codes hard to understand. Therefore, we back this change out and still use Space line to distinct each part of codes.

3. Prest Metrics:

We first compare the Prest Metrics between the original "UserInterface" Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	0.02	247	210	227	78	20	25	437	2890.6
Refactored	0.01	148	212	207	102	21	20	419	2908.9

Since we mainly refactor the method "showDataForYear()", we then compare the Prest Metrics between the original method and the refactored method.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	5.33	16	16	20	14	7	4	36	158.1
Refactored	5.33	16	12	16	10	7	4.17	28	114.4

4. Improved Prest Metrics

Clearly, in the method comparison, the metrics of the refactored method Main() are almost better than the original method.

(1)Length of code: Since this method is very short, all we've done is to change the variable names to make it clearer. So the length remains the same.

(2)Operators & Operands: both operators and operands (except the unique operator count) has been reduced, which would make the refactored method is easier to read and understand.

(3)Halstead metrics: clearly, through refactoring, the Halstead metrics is improved, which also indicates the refactored method is better than the original method in readability.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of the refactored class have gone a little worse.

This is due to we introduce many new small method into the refactored class while the original method only have 3 methods. Therefore the total complexity seems to be increased. However, since the functionality of all the small method is clear enough, the readability is not deteriorated in fact. Therefore, it may not be appropriate to evaluate the refactoring based on the whole class in our case.

showDataForRange() Method

1. Code Smells & Solutions:

(1)Line 120-122 & 143-148: These lines of codes is to output the results, which are not very related to the functionality of the original method and causes the Long Method problem;
Solution: We create a new method "ouputRange()" to extract these code, thus making it easier to read;

(2)Line 129: The variable "x" is unclearly defined. Besides, in this method, it's a redundant variable. Since we could judge whether the result is empty by other means without introducing extra variables;
Solution: We eliminated the variable "x" and in line 143, we use "result.toString().equals("")" to replace "x". This could reduce the redundancy and make the codes more clear;

(3)Line 124-132: These two lines of codes are not cohesive with the functionality of the method, thus it's desirable to split them into different methods;
Solution: We create a new method "GetList()" to split them up. Besides, this method is also used in other methods which try to load the WorldSeries instances from the data;

(4)Line 133: The variable "wsi" is unclearly defined;
Solution: We change its name to be "instance";

(5)Line 136-137: These two lines are to add the result with suitable instances, which involves too much "knowledge" to understand, such as "year()", "winner()" etc. Therefore, it's a Message Chain problem, or also a Long method problem.
Solution: We create a new method "UpdateResult()" to hide the process in the original method, thus making it easier to understand.

2. Removed Changes:

We originally planned to extract the "if condition" in line 134, which may be regarded as a Message Chain problem. However, then we believe that it's more nature and straightforward to leave it unchanged. Since it's very easy to understand what the functionality of this part is in the method.

3. Prest Metrics

We first compare the Prest Metrics between the original "UserInterface" Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	0.02	247	210	227	78	20	25	437	2890.6
Refactored	0.01	148	212	207	102	21	20	419	2908.9

We then compare the Prest Metrics between the original "showDataForRange()" method and the refactored method.

	Cyclomatic_C	Lines_of_	Total_	Total_	Unique_oper	Unique_ope	Halstead_	Halstead_l	Halstead_

	omplexity	code	operands	operators	ands_count	rator_count	difficulty	ength	volumn
Orignal	6	33	23	35	18	13	8.33	58	287.3
Refactored	3.25	13	10	13	8	8	5	23	92

4. Improved Prest Metrics:

Although the original method is not very hard to understand, through refactoring, we can still see the obvious improvements in the readability of the refactored one. Since all the metrics is improved a lot.

(1)Cyclomatic_complexity: we decrease the cyclomatic_complexity metrics form 6 to 4.

(2)Length of code: the length of code is reduced to 13 compared to the original 33, which is really much reduction considered the original method is small, also showing the effectiveness of our refactoring means;

(3)Operators & Operands: both operators and operands have been reduced, which is also the reason why the Halstead metrics could be improved;

(4)Halstead metrics: clearly, through refactoring, the operators and operands have been reduced. Therefore, the Halstead metrics is improved a lot. The Halsted_volumn is reduced from 287.3 to 92.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of the refactored class have gone a little worse.

This is due to we introduce many new small method into the refactored class while the original method only have 3 methods. Therefore the total complexity seems to be increased. However, since the functionality of all the small method is clear enough, the readability is not deteriorated in fact. Therefore, it may not be appropriate to evaluate the refactoring based on the whole class in our case.

showDataForTeam() Method

1. Code Smells & Solutions

(1) **Lines 160-182, 187-207 and 213-239**: **Duplicate Code& Long Method**.Generally, three"if-else" conditions in method showDataForTeam(), namely, are apparently code smell of Long Method. In addition, they are duplicate code, although their forms differ slightly, three parts do essentially the same thing: search results from all the data, and append the search results to "result." Object.
Solution: Extract Method: To address the Duplicate Code Problem, we use Extract Method technique and create a new method called appendAllResults(), which take the orginal data, Stringbuffer "result", the choice of user, and team name, as inputs, and append the "result" object with all the search results.

(2) Lines 175-182, 200-207 and 232-239 :**Duplicate Code& Long Method**: After step (1), we find that the "if-else" are not coherent to the previous method. Plus, they are apparently duplicate code. These three parts do essentially the same thing: append the general result to StringBuffer object "result" to show how many games. totally the given team has won/lost.
Solution:Extract Method: Similarly, we use Extract Method to create a new method called appendGenralInfo(), which tells the users how many games totally the given team has won/lost. It takes Stringbuffer "result", team name, "wonOrLost "selection, number of won, and number of lost as inputs, and append the general information to "result" object.

(3) **Line 167, 193, 219 and 225**: **Message Chain:& Duplicate Code**. After these general modification, we find that the condition is too long. In order to know if we have found the incidence in the data, we have to get through a lot of inner steps, like winner()->toUpperCase().contains and etc. This is a code smell of Message Chain. Besides, there are a lot of duplicate message chains, that make things worse.
Solution: Hide Delegate &Extract Method. We use a new method called containTeam() , then go through each of the steps, so that the users do not have to know the inner scheme of the finding condition.

(4) **Line 154,164,190,216**: **Naming**: After these big modification, we then find that some of namings are not so clear defined. First, in line 154, "ds" is not clear what this variable represents.
Solution: Let's change "ds" to "allData" to indicate all the data from DATAFILE.

(5) **Line 164,165**: **Naming**: Similarly, "list" is not clear what this variable represents.
Solution: Let's change "list" to "allIncidencesList" to indicate all the data from DATAFILE.

(6) **Line 165,167,169,191 and etc**: **Naming**: Similarly, "wsi" is not clear what this variable represents.
Solution: Let's change "ds" to "oneIncidence" to indicate one of the incidences in "allIncidenceList"

(7) **Line 161,187,213**: **Naming, Duplicate Code**: Similarly, "m" ,"a" and "b" are not clear what these variable represents.
Solution: Let's change "a" to "countWon" , "b" to "countLoss", "m" to"countWon + countLoss", to indicate the number of wining/losing games in one query.

2. Removed Changes:

Since we can simply decrease the value of metric “lines of codes” by eliminating the Space line, we tried this first to make the improvement. However, in fact, eliminating the Space line will decrease the readability of codes, thus making codes hard to understand. Therefore, we back this change out and still use Space line to distinct each part of codes.

3. Prest Metrics:

We first compare the Prest Metrics between the original “UserInterface” Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	0.02	247	210	227	78	20	25	437	2890.6
Refactored	0.01	248	212	207	102	21	20	419	2908.9

Since we mainly refactor the method “showDataForTeam()”, we then compare the Prest Metrics between the original method and the refactored method.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	7	98	117	126	35	9	14.29	243	1326.6
Refactored	5.75	23	24	14	12	6	5.88	38	158.5

4. Improved Prest Metrics

Clearly, in the method comparison, the metrics of the refactored method are almost better than the original method.

(1)Cyclomatic_complexity: we decrease the cyclomatic_complexity metrics form 7 to 5.75.

(2)Length of code: the length of code is reduced to 23 compared to the original 98, which is really much reduction considered the original method is small, also showing the effectiveness of our refactoring means;

(3)Operators & Operands: both operators and operands have been reduced, which is also the reason why the Halstead metrics could be improved;

(4)Halstead metrics: clearly, through refactoring, the operators and operands have been reduced. Therefore, the Halstead metrics is improved a lot. The Halsted_volumn is reduced from 1326.6 to 158.5.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of the refactored class have gone a little worse.

This is due to we introduce many new small method into the refactored class while the original method only have 4 methods. Therefore the total complexity seems to be increased. However, since the functionality of all the small method is clear enough, the readability is not deteriorated in fact. Therefore, it may not be appropriate to evaluate the refactoring based on the whole class in our case.

“DataStore” Class

readDataFile() Method

1. Code Smells & Solutions:

(1)Line 8: This Boolean variable is defined but not used, thus is a redundant variable;
Solution: We simply delete this variable;

(2)Line 11, 12: This 4 variables “y”, ”w”, ”l” and ”s” is unclearly defined and hard to understand;
Solution: We redefined the name of these variables as “year”, “winner”, “loser” and “score” to make it meaningful;

(3)Line 31-40: These lines of codes is the process of obtaining each element of one WorldSeries instance, which is not very cohesive with the functionality of the “readDataFile()” method, thus making the method a Long method;
Solution: We split this part of codes into a new method called “ReturnInstance()”, which makes the “readDataFile()” method easier to read and understand;

(4)Line 34-36: This is a Message Chain problem. Since too many need to be understand to obtain the value of “loser”;
Solution: We use Hide Delegate that create a new method “GetLoser()” in order to get “loser”, therefore we can ignore the process of how to get “loser” in the “readDataFile()” method;

(5)Line 40: The variable “wsi” is defined unclearly;
Solution: We change its name to be “instance”, which is easier to understand.

2. Removed Changes:

Since we can simply decrease the value of metric “lines of codes” by eliminating the Space line, we tried this first to make the improvement. However, in fact, eliminating the Space line will decrease the readability of codes, thus making codes hard to understand. Therefore, we back this change out and still use Space line to distinct each part of codes.

3. Prest Metrics:

We first compare the Prest Metrics between the original “Datastore” Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	13	45	24	16	14	6	5.26	40	172.9
Refactored	14	41	27	19	14	6	5.88	46	198.8

Since we mainly refactor the method “readDataFile()”, we then compare the Prest Metrics between the original method and the refactored method.

	Cyclomatic_C	Lines_of_	Total_	Total_	Unique_oper	Unique_ope	Halstead_	Halstead_l	Halstead_

	omplexity	code	operands	operators	ands_count	rator_count	difficulty	ength	volumn
Orignal	3	33	18	11	11	4	3.23	29	113.3
Refactored	3	13	6	5	5	5	3.03	11	36.5

4. Improved Prest Metrics

Clearly, in the method comparison, the metrics of the refactored method Main() are almost better than the original method.

(1)Length of code: since we extract many uncorrelated codes from the original method, the length of the refactored is improved largely. This has solved the Long Method problem;

(2)Operators & Operands: both operators and operands (except the unique operator count) has been reduced, which would make the refactored method is easier to read and understand.

(3)Halstead metrics: clearly, through refactoring, the Halstead metrics is improved a lot, which also indicates the refactored method is better than the original method in readability.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of the refactored class have gone a little worse.

This is due to we introduce many new small method into the refactored class while the original method only have some methods. Therefore the total complexity seems to be increased. However, since the functionality of all the small method is clear enough, the readability is not deteriorated in fact. Therefore, it may not be appropriate to evaluate the refactoring based on the whole class in our case.

“Sorter” Class:

GetWinner Method

1. Code Smells & Solutions:

(1)Line 9: The name of method "winners()" is unclear and may be confused with the method "winner" in "WorldSeriesInstance()" class;
Solution: We change the method's name to be "GetWinners()";

(2)Line 10: The variable "ds" is unclearly defined;
Solution: We change its name to be "data";

(3)Line 10 & 15: These two lines of codes are not cohesive with the functionality of the method, thus it's desirable to split them into different methods;
Solution: We create a new method "GetList()" to split them up;

(4)Line 13 & 77-84: It's the same problem: Long method. Since these lines' function is to output the result, they should be split into a new method. Besides, line 79-84 is to output the years of a specific winner team, which may also cause the decrease of readability of the new created method;
Solution: We create two new methods "output()" and "outputYears()" to contain these lines of codes respectively, thus making the "GetWinners()" easier to understand;

(5)Line 27 & 28: The two variable "teams" and "wins" are unclearly defined. Actually, we may not identify what they represent and misunderstand from each other;
Solution: We change their name to be "teamNames" and "winYears";

(6)Line 27 & 28: Since we split the original Long method into many small pieces of short methods, the two variable "teamNames" and "winYears" are frequently used in many of the short methods, we think it's necessary to make the two variables to become the field of this class. Additionally, in common sense, these two variables are closely related the functionality of this class "Sorter", which also enhances our confidence;
Solution: We change the two variables to become the fields of this class;

(7)Line 30: The variable "wsi" is unclearly defined;
Solution: We change its name to be "instance";

(8)Line 31-48: This part of code is to justify whether the given instance's name is new or not. Since it causes too much to understand in the original method and lead to Long Method, thus it should be extracted;
Solution: We create a new method "NewNameOrNot()" to extract this part of code;

(9)Line 54-74: This part of codes is to sort the sequence of the ArrayList "teamNames" and "winYears" in alphabetical order, which is not cohesive with the functionality of the original method and leads to the Long Method problem;
Solution: We create a new method "sortData()" to extract this part of codes;

(10)Line 56-57: "min" and "minIndex" are defined inside the Loop, while we think it's more straightforward to define them outside the Loop to indicate the functionality of this part;
Solution: We redefined the two variables outside the Loop in our new method "sortData()";

(11)Line 59: This line leads to the Message Chain problem to some extent, although it's not very severe;

Solution: We create a new method “CompareNames()” to hide this process, thus making the new method easier to read;

(12)Line 65-67 & 70-72: These two parts of codes are to swap elements in ArrayList “teamNames” and “winYears”, which causes the Long Method problem;
Solution: We create two new methods “SwapNames()” and “SwapYears()” to extract these two parts of codes, thus making it easier to understand;

(13)Line 66: This line uses “teams.get(minIndex)” to get the team name of the minIndex. While in fact this name is already obtain in the variable “min”. Therefore this causes the redundancy;
Solution: We replace “teams.get(minIndex)” with “min”.

2. Removed Changes

The first removed change is the same as the “DataStore” class: we eliminate all the Space line to decrease lines of code. However, in the last we still keep the space line to keep the readability of codes;
The second removed change is that we eliminated the method “GetList()”. Since in the “UserInterface” class, we have created another new method “GetList()” which has the same functionality as that in “Sorter” class. Therefore we removed the redundant method “GetList()” in “Sorter” class, because we believe the method “GetList()” is more closely related with the functionality of “UserInterface” class.

3. Prest Metrics

We first compare the Prest Metrics between the original “Sorter” Class and the refactored Class.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	10	82	56	48	20	11	16.67	104	515.2
Refactored	26	81	71	57	29	11	14.29	128	681.2

We then compare the Prest Metrics between the original “winners()” method and the refactored “GetWinners()” method.

	Cyclomatic_Complexity	Lines_of_code	Total_operands	Total_operators	Unique_operands_count	Unique_operator_count	Halstead_difficulty	Halstead_length	Halstead_volumn
Orignal	9	81	56	48	20	11	16.67	104	515.2
Refactored	2	11	7	7	7	4	2	14	48.4

4. Improved Prest Metrics:

Clearly, in the method comparison, the metrics of the refactored method are much better than the original method. We extract many codes which are not very related to the “GetWinners()” method and solve both the Long method problem and Message Chain problem. Besides, we rename many meaningless variables to make the refactored method easier to understand. It’s not surprising that the whole metrics is improved a lot.
(1)Cyclomatic_complexity: we decrease the cyclomatic_complexity metrics a lot from 9 to 2.
(2)Length of code: the length of code is reduced to 11 compared to the original 81, which is really much reduction, also indicating the effectiveness of our refactoring means;
(3)Operators & Operands: both operators and operands have been reduced largely, which would make the refactored method is easier to read and understand.
(4)Halstead metrics: clearly, through refactoring, the Halstead metrics is improved a lot. Since there is only one

method in the original class while we create many new small method to increase readability, the Halsted_volumn is improved a lot from 515.2 to 48.4.

5. Deteriorated Prest Metrics

In the class comparison, the metrics of our refactored class is slightly deteriorated compared to the original class, which is the same as that in “DataStore” class. The reason may be that in original class there is only one method. Although the readability of that method is very low, the class seems to be “clear” enough; while in the refactored class there are many small methods whose functionality is fairly easy to understand, but in class view, the total complexity increases with the increase of numbers of methods.

However, the length of codes of the refactored class is equal to the original method, which means the added new methods don’t introduce redundancy. Besides, the Halstead_difficulity is improved in the refactored method, also showing the effectiveness of our refactoring.