**UNIT 14 ASSASSIGNMENT 1 –**

**POND-U-LIKE PLUS**

**P3.**

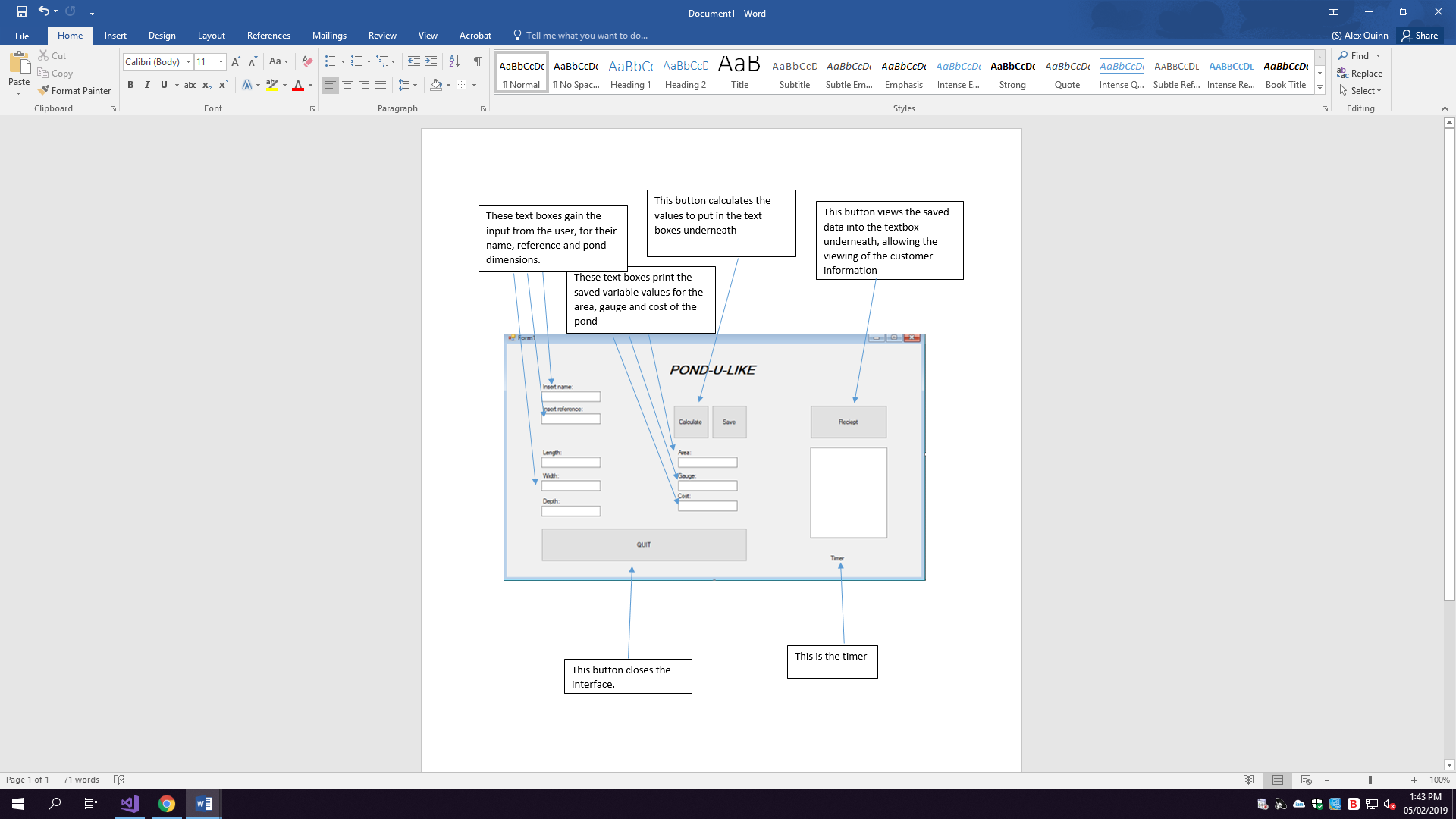
**Purpose of program:**

The program must be able to receive the inputs for the length, width and depth of the pond of the customer, while having validation checks for the inputs to ensure that they fit the parameters set by the specification. The program must also receive inputs for the user’s name and reference number.

Once these inputs have been received, the area must be calculated, along with the gauge and cost of the liner for the pond, followed by the displaying of this on the programs interface.

The data must then be saved into a text file, being: **Customer reference, Customer name, pond area, cost and liner.** This data must also be retrieved and displayed.

In addition to all of this, the current time and date must be included on the form.

**INTERFACE DESIGN**

**DATA DICTIONARY:**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Data Type** | **Purpose** |
| **CName** | String | The customer’s name will be best suited to the string variable type due to the fact that it is a combination of characters |
| **Refnum** | Integer | The reference number will be a whole number, because of this, integer would best suit the value |
| **Length** | Decimal | The length will be a value less than 10, but may not be a whole number, therefore using a decimal data type would make it easier to store. |
| **Width1** | Decimal | The width will be a value less than 10, but may not be a whole number, therefore using a decimal data type would make it easier to store. |
| **Depth** | Decimal | The depth will not be larger than 2, so again decimal would best suit this as it will be a small, but potentially not a whole, number. |
| **Area** | Decimal | The area will be a result of the length, width and depth, so it may not be a whole number and would therefore best suit the type of decimal. |
| **Gauge1** | Constant | The value of gauge one will remain at 1.12 due to the fact that, 1.12 will be the multiplier to calculate the cost if the area doesn’t meet gauge 2 requirements. |
| **Gauge2** | Constant | The value of gauge two will remain at 1.76 as this will be the multiplier to calculate the cost if the area exceeds gauge 1 requirements. |
| **Gaugeset** | String | Depending on what gauge is used, gaugeset will be assigned to “gauge 1” or “gauge 2” and will be printed in the corresponding textbox, as it is a word, string would be best |
| **Cost** | Decimal | The price may not be a whole number, so this data type will suit it best. |
| **Errorflag** | Boolean | This will be either true or false to determine whether the inputs are correct and will act as a validation method. |
| **Area1** | Decimal | The area may not be a whole number, therefore a decimal would be the best data type to store it. |
| **Temparea** | Decimal | As this is used to calculate the area within the custom function, decimal would be the best data type again. |
| **Answer** | Decimal | Answer would be a decimal as it is used again within the area calculation. |

**Pseudocode:**

Receive name input

Receive reference input

Receive length, width and depth inputs

Check length, width and depth against error flag

If the parameters are not met, display the error message

If the parameters are met, calculate (length + width + (depth x 4) and display the area within the box

Check the parameters to display either gauge 1 or gauge 2

If gauge 1, display the price as (area x 1.12) or if gauge 2 display price as (area x 1.76)

When save is clicked

Open text file

Save data

Close file

When view button is clicked

Open file

Loop while displayed

Display text

Loop until closed

Close file

END

**Event Diagrams:**

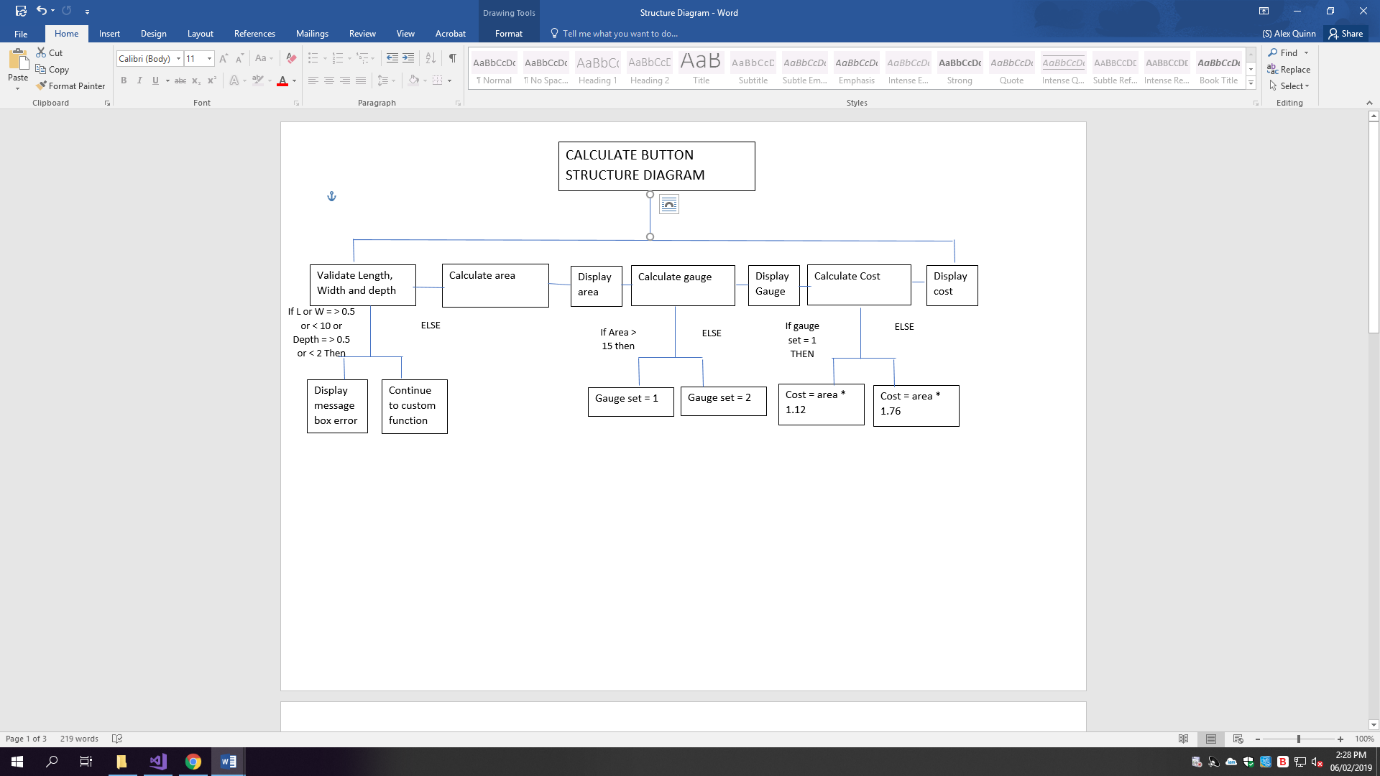
|  |
| --- |
| Event: **BUTTON** |
| **Click event:**  When the button is clicked, the inputs given will be used to calculate the necessary values, once they have been calculated, they will be placed into the corresponding fields within the user interface. |
| **CODE:** |

|  |
| --- |
| Event: **Button** |
| **Click event:**  When clicked, the button will save the data to a text file, which will be then keep a second copy of the form once it is filled out, to be retrieved if necessary. |
| **CODE:** |

|  |
| --- |
| Event: **Button** |
| **Click event:**  When clicked, the button will retrieve and view the data that was saved from the “save” button, into the underneath textbox for the user to view. |
| **CODE:** |

|  |
| --- |
| Event: **Timer** |
| Event: **Time driven:**  When the programme starts, it will display the date and time on the interface for the user. |
| **CODE:** |

**Structure Diagrams:**





**P4 – The Code.**

Public Class txtdisplay

'These are the global variables I will use for the code.

Dim cname As String

Dim reference As Single

Dim length As Single

Dim width1 As Single

Dim depth As Single

Dim area As Single

'The Const function keeps the same value throughout.

Const gauge1 = 1.12

Const gauge2 = 1.76

Dim gaugeset As String

Dim cost As Decimal

Dim errorflag As Boolean

Dim answer As Decimal

Private Sub btncalc\_Click(sender As Object, e As EventArgs) Handles btncalc.Click

'These variables set the variable values.

length = txtlength.Text

Width = txtwidth.Text

depth = txtdepth.Text

'The errorflag will be used to validate the values input to the variables, when false, or when the numbers are incorrect, an error will appear.

errorflag = True

If errorflag = False Then

MsgBox("The value input is invalid")

End If

'These IF statements validate athe input measurements to ensure they meet the specification

If length < 0.5 Or length > 10 Then

errorflag = False

If Width < 0.5 Or width1 > 10 Then

errorflag = False

If depth < 0.5 Or depth > 2 Then

errorflag = False

End If

End If

End If

'This is setting the custom function to calculate the area.

area = area1(length, width1, depth)

txtpondarea.Text = area

'These IF statements allow for the gauge to be chosen in response the calculated area.

If area < 15 Then gaugeset = "Gauge 1"

If area > 15 Then gaugeset = "Gauge 2"

txtgauge.Text = gaugeset

'These IF statements work out the cost, corresponding with the gauge needed, being gauge1 or gauge2 depending on the area.

If area < 15 Then cost = area \* gauge1

If area > 15 Then cost = area \* gauge2

txtcost.Text = cost

End Sub

'This function is used to calculate the area which was previously referred to.

'The l, w and d are the values within the variables, which are used within the formula so it can be used multiple times.

Function area1(ByVal l As Decimal, ByVal w As Decimal, ByVal d As Decimal) As Decimal

'This variable is the temporary area, that is given as the answer from the function

Dim temparea As Decimal

'This is the formula used to calculate the area within the custom function.

temparea = (l + (d \* 2)) \* (w + (d \* 2))

'This returns the temparea value as the functions value, making the varibale of area = the answer given.

Return temparea

End Function

Private Sub btnsave\_Click(sender As Object, e As EventArgs) Handles btnsave.Click

cname = txtname.Text

reference = txtrefnum.Text

'This creates or opens the text file, and adds the necessary fields to the file from the user interface and stored variables, The file is then closed and the data saved.

FileOpen(1, "Saved data.txt", OpenMode.Append)

PrintLine(1, "Name: ", cname, "& Reference: ", reference, "& Pond area: ", area, "& Pond cost: £", cost, "& Gauge: ", gaugeset)

FileClose(1)

End Sub

Private Sub btnreciept\_Click(sender As Object, e As EventArgs) Handles btnreciept.Click

'This opens the file, and while it is open, the field for the viewing of the data is set to the contents of the file, allowing for it to be displayed.

FileOpen(1, "Saved data.txt", OpenMode.Input)

While Not EOF(1)

txtreciept.Text = LineInput(1)

End While

End Sub

Private Sub Timer1\_Tick(sender As Object, e As EventArgs) Handles Timer1.Tick

'This allows for a label to be turned into a timer, which displays the current date and time when in use

lbtime.Text = Now()

End Sub

Private Sub txtquit\_Click(sender As Object, e As EventArgs) Handles txtquit.Click

'This is for the quit button and closes the program.

Close()

End Sub

End Class

**P5 – TESTING**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **TEST** | **DATA USED** | **EXPECTED RESULT** | **ACTUAL RESULT** | **PASS/FAIL** | **IMPROVEMENT(IF NECESSARY)** |
| To test whether the name can be input | “Alex” | For the name to be within the text box |  | PASS |  |
| To test whether the reference can be input | 123 | For the number to within the textbox |  | PASS |  |
| To test whether the length can be input | 1 | For the length to be within the length box |  | PASS |  |
| To test whether the width can be input | 1 | For the width to be input within the textbox |  | PASS |  |
| To test whether the depth can be input | 1 | For the number to be within the textbox |  | PASS |  |
| To test whether the area is displayed correctly (L + W + (D \* 4) | 1,1,1 | For the area to display as 9 |  | PASS |  |
| To test whether the gauge displays correctly | 1,1,1 | For the gauge to be 1 because the area is less than 15 |  | PASS |  |
| To test whether the cost displays correctly | 1,1,1 | For the cost to be 6.72 |  | PASS |  |
| To check whether the data can be saved | 1,1,1 | For the data to be saved in a text file. |  | PASS |  |
| To check whether the data is viewed in the text box from the file | 1,1,1, | For the data to display within the textbox |  | PASS |  |
| To check whether the timer is displayed | N/A |  |  | PASS |  |
| To test the lower extreme boundary | 0.4, 0.5, 0.5 for each value | For the calculation to carry out |  | PASS |  |
| To test the higher boundary. | 10.1, 10, 1 | For the calculation to display the upper boundary |  | PASS |  |
| To test whether the error message appears when data is incorrect | 11,10,1 | For the message to display | To fix this I defined my errorflag as FALSE when initialising the variable, this fixed the error flag issue and displayed the error message | FAIL |  |
| To test whether the wrong length input displays error | 11, 10 , 1 | For the error message to display |  | PASS |  |
| To test whether the wrong width input displays error | 10, 11, 1 | For the error message to display |  | PASS |  |
| To test whether the wrong depth input displays error | 10, 10, 3 | For the error message to display |  | PASS |  |

**TASK P2 & M2**

|  |  |  |
| --- | --- | --- |
| **Code** | **Description (p2)** | **Reason for use (m2)** |
|  | **Control:** The controls are features such as buttons or text boxes. They allow input from the user or allow algorithms to be executed when clicked, while also allowing for a basic structure to be created for the interface  **Control properties:**  Control properties are used to edit or modify the controls within the programme; this could be through editing the text box to be multi line to allow the text file to be viewed. | **Control:** I have chosen to use controls to allow the user to input data for the code to be executed with. The controls correspond with variables to compile the code, the text boxes are for values and answers while the final, larger text box is to view the text file once it has been saved. They allow the code to run on the inputs given while also allowing for the triggers.  **Control properties:**  The properties are used to edit and modify the controls. For example, I have used the editing tool to rename the text boxes to best suit the variable names while also changing the size of the final text box to make it multi lined and with scroll bars. The properties allow for the ease of coding while also allowing aesthetic changes. |
|  | **Event handler:**  The code, which programmers write to make an event respond in the intended way. E.g. saving the text to a file when the save button is clicked. This code saves the text to the text file. However, an event handler could be any piece of code of which results in an outcome. | **Event Handler:**  I have used this to save the text, which is given from the calculations in to a text file to meet the specification. This event handler is also working alongside a button. This leads to the event of the text saving and forces the code to work in a particular but intended way. Other examples would be a calculation within a custom function which would make the code calculate the value on the event of a button click |
|  | **Trigger - Left mouse click.**  Something which a user, another program or system does to cause an event e.g. pressing a releasing the left mouse button. This trigger is something that leads to an event. | **Trigger:** In this instance, I have used the trigger of a button to calculate the area, gauge and cost. Upon the pressing of the button, the trigger will be the event of the calculations. The event that occurs will depend on what event has been carried out. Another example of a trigger in this program is the save button which begins the event of saving the text. The receipt button begins the event of displaying the text. |
|  | **Event – Mouse click and time**  The response to a trigger, e.g. a click event fires if the correct trigger is detected  This could be the event of calculating the values for area, cost and gauge. While also being the event of a clock being displayed on the program, as a time driven event. This can be seen with the clock at the bottom of this screen shot. | **Mouse click:**  This mouse click event displays the receipt within the text box because of the button trigger. I have chosen to use this event because the specification asks for the viewing of the data once it has been input. The viewing of the text is the response to the button being clicked. Another example could be the event of clicking the calculate button, bringing the event of displaying the data into action.  **Time:**  The time driven event occurs as soon as the program opens. The event would be automatic, would display once the program opens, and would continue until the program closes. Displaying the current date and time. |
|  | **Custom function and parameters –**  A custom function is a piece of code, including parameters used to pass a value through to a specific area. The function can be called upon multiple times in various scenarios. The parameters set, mean only specific data types can be used within the function for the result to be passed on. | **To pass a value through –**  I chose to use a custom function for my area calculation; this is due to the fact that the value can be calculated several times and so the calculation can be called upon when it is needed. I have also chosen to use it because a value must be passed to give the area figure, and the parameters allow for this figure to be sent through as a variable. |
|  | **A procedure (a button)**  A procedure allows for the code to be modular, allowing each procedure to be completed step by step. | **To make the code modular –**  By making the code modular, it allows for the program to break the code down part by part, opposed to running all of it simultaneously, meaning that it would have a modular structure to run in the correct order. |

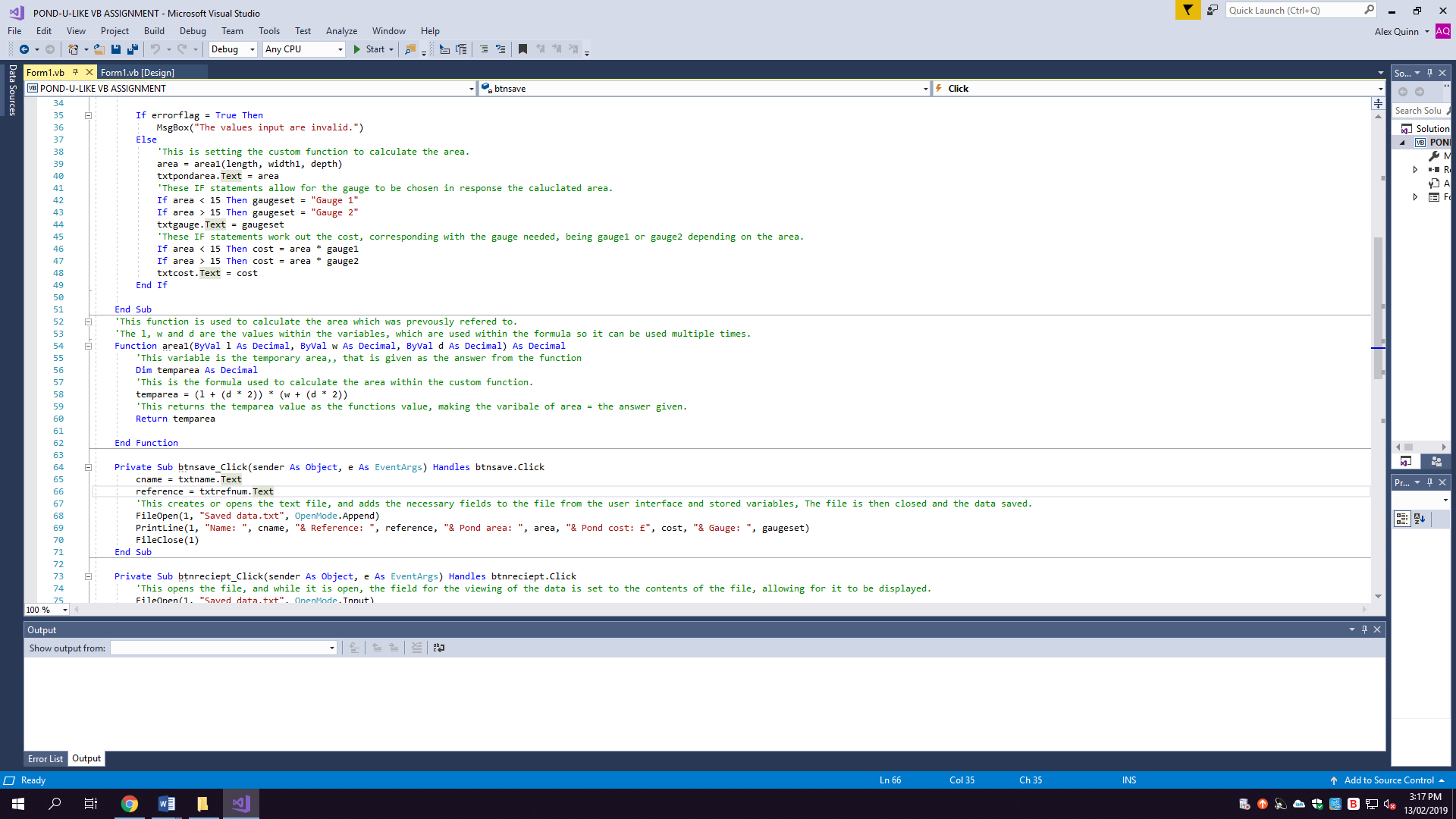
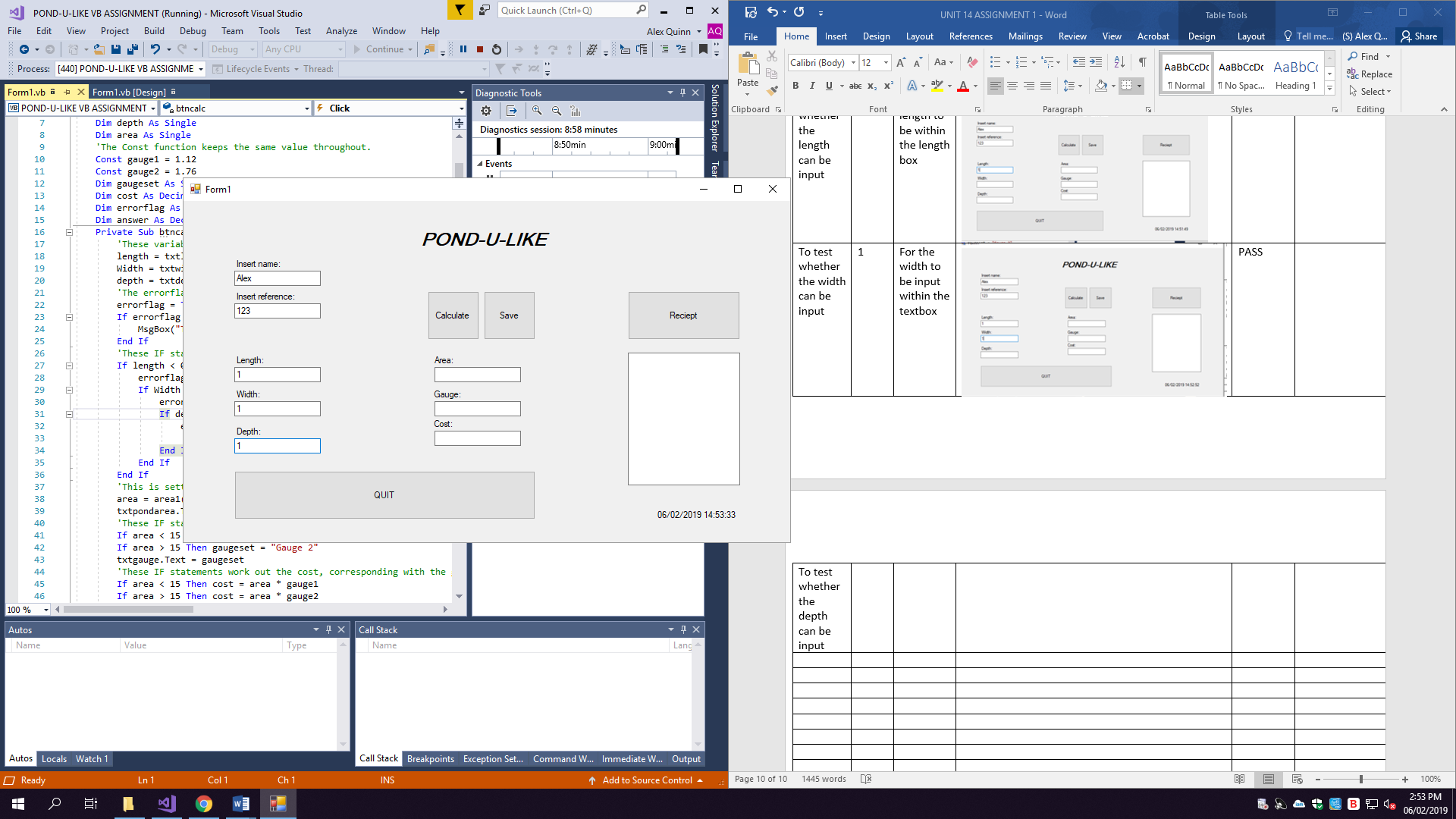
**TASK 6 – D2**

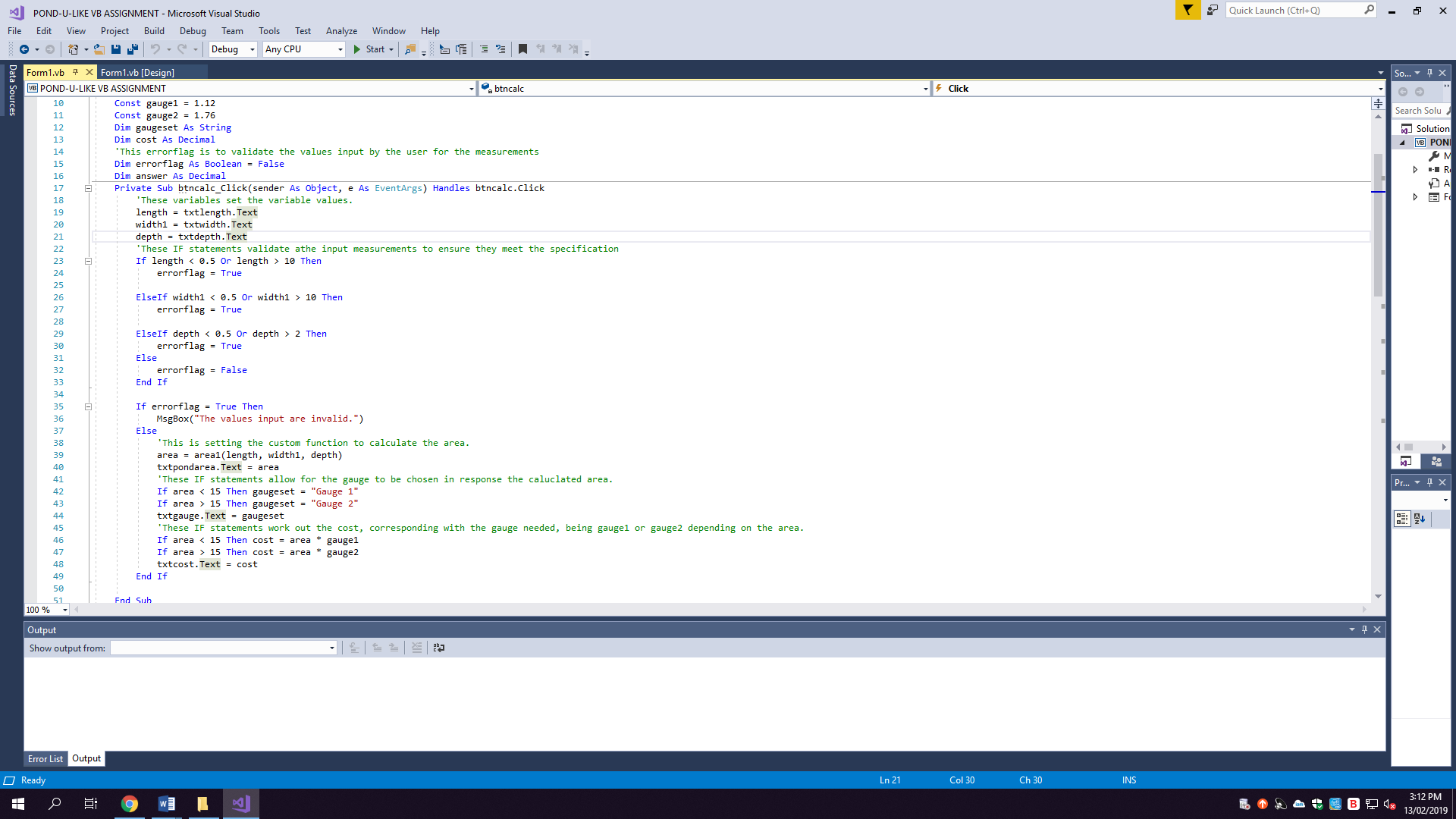
**Discussing the degree of success, and failure:**

**Inputting the length, width, depth, name and reference**

The user can input their name and reference number (Tests 1, 2 support this) into the text boxes, while then being able to input their three values separately (Tests 3, 4, 5 support this) into the user interface. These are then stored to to temporary variales to be used in calculations or to store in the text file. They can be changed if the program is used multiple times. Through using the variables it allows for the values to be stored throughout the time the progrma is open for the rest of the code to execute. If this was not the case, no values would be used in the following tasks and the program would not work. This meets the criteria because it gives a location to store the necessary data withint the code.

As you can see here, the input numbers and name would be stored in the variable corresponding to whichever value is given. This means that whenever the variable is used, the value that is stored within it is assigned to the scenario to be used in whichever instance.

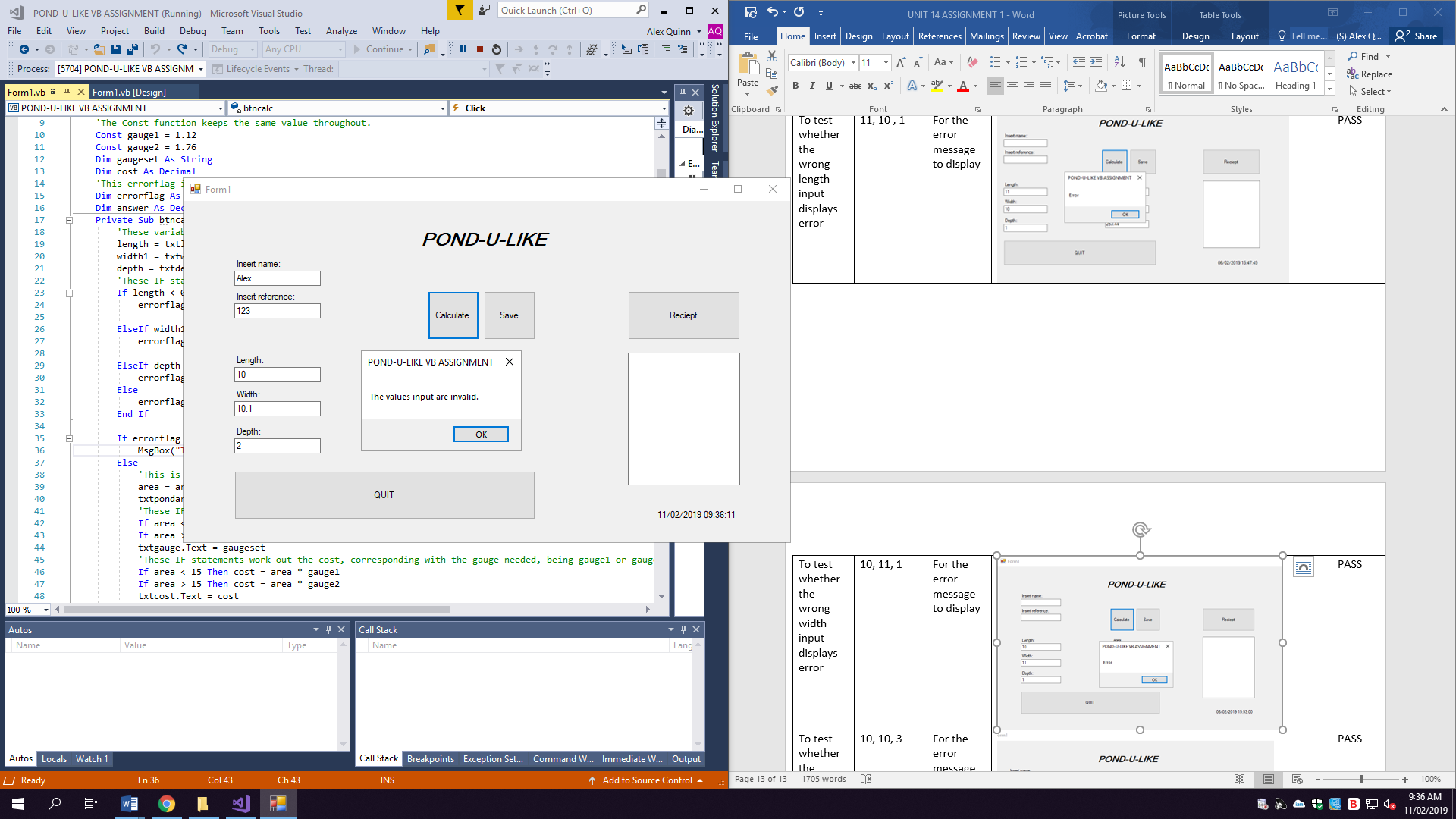
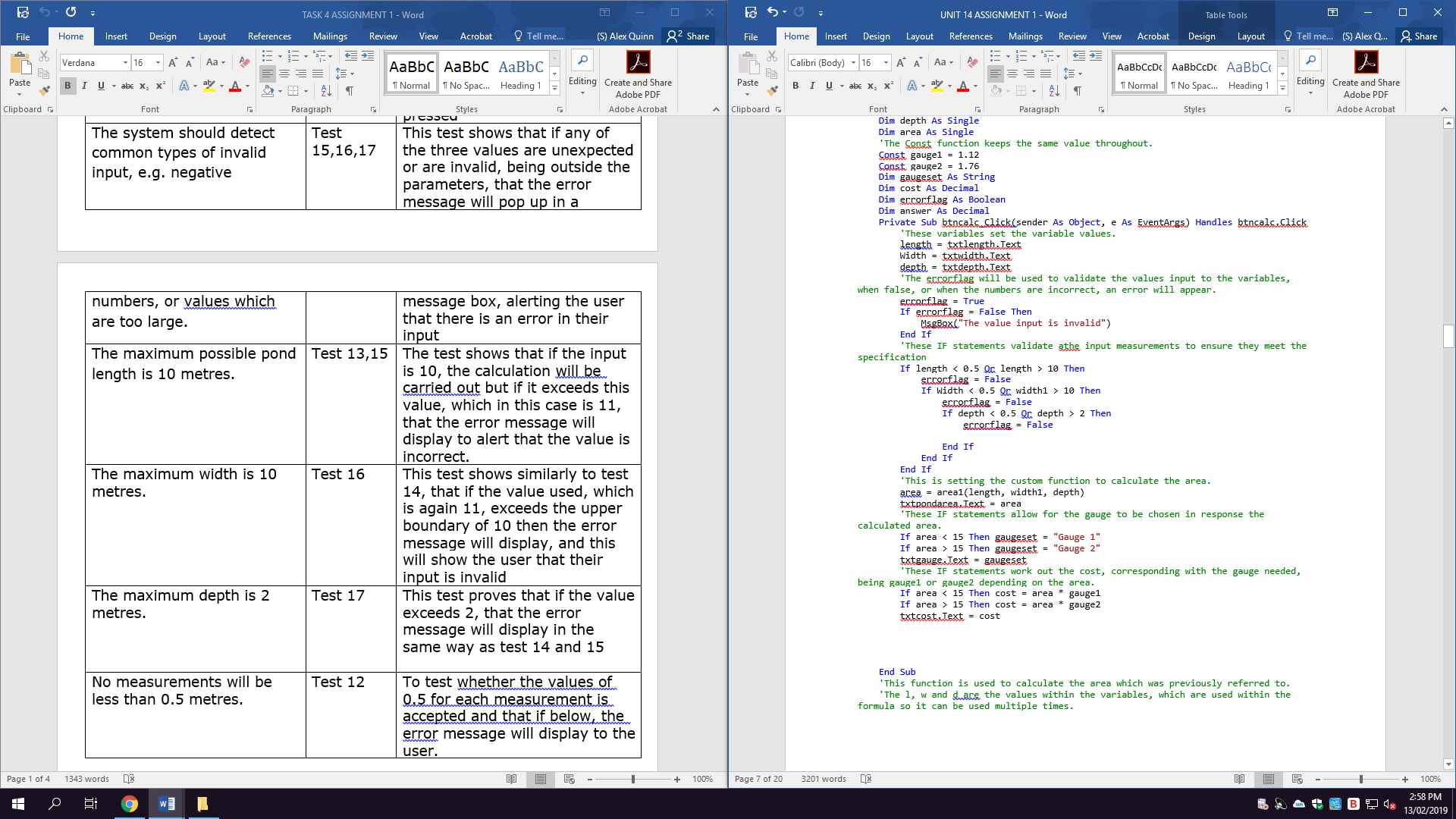




**Have validation to check incorrect inputs:**

I have used an error flag within my code to make sure that the input measurements fit the maximum and minimum lengths. This allows the code to pick up any errors in the input, for example, if the user were to input a length of 12, an error would display to show the user that there has been an error in the input. There are also error flags for the width and depth to achieve the same goal. The error flag ultimately makes sure that all data used within the code fits the requirements for the calculations to work out the gauge and price needed for the liner. Without the error flag the code would accept any value for the measurements but this wouldn’t suit the requirements due to the fact that Pond-U-Like have stated the maximum and minimum requirements for the liner. (This is supported in tests 12 to 17)

As you can see, the validation ensures that the value of 10.1 is not accepted within the code, because it is not within the specific measurements given within the specification. However, if the value were between 0.5 and 10 it would be passed into the area and gauge calculation.

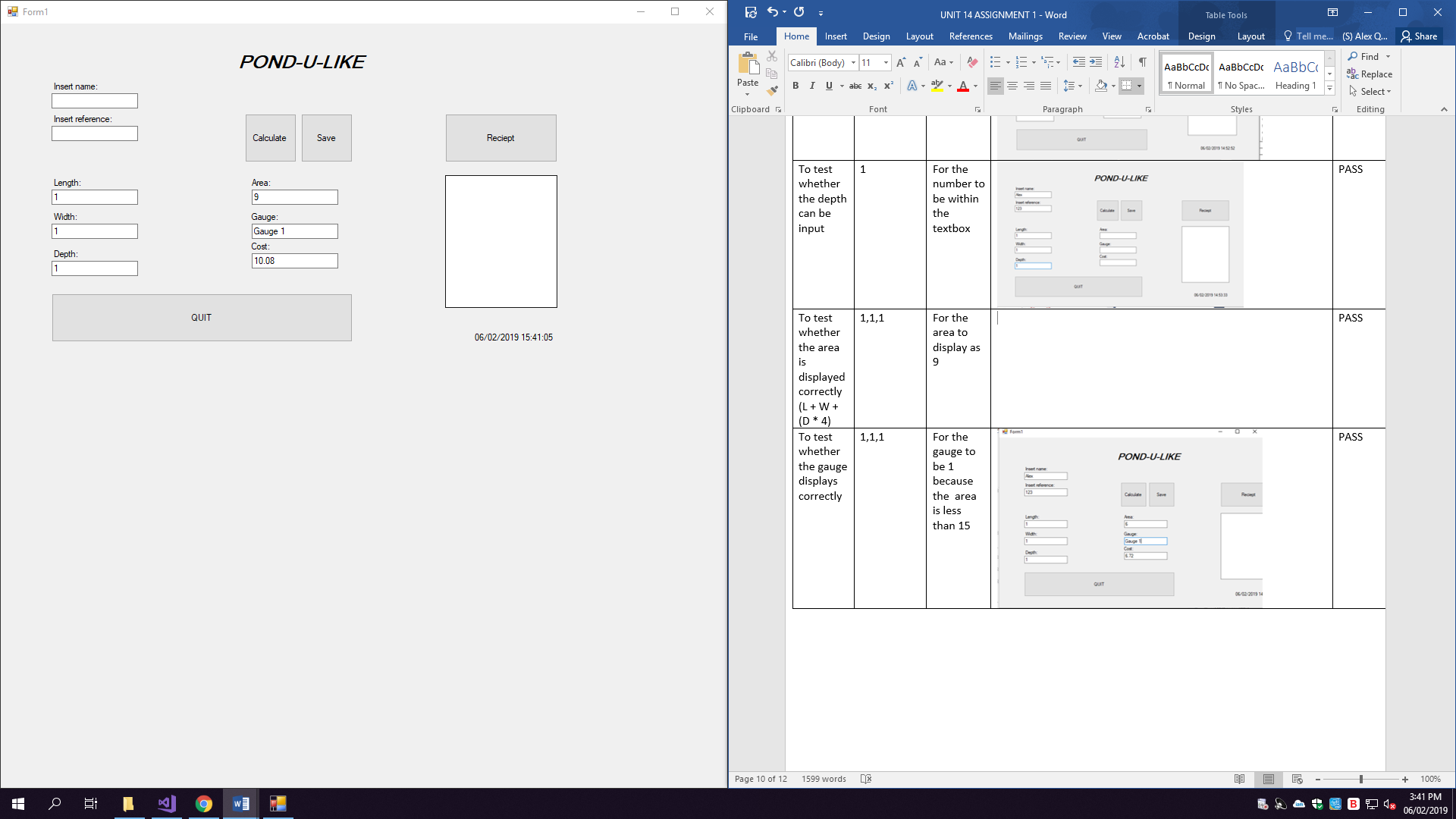


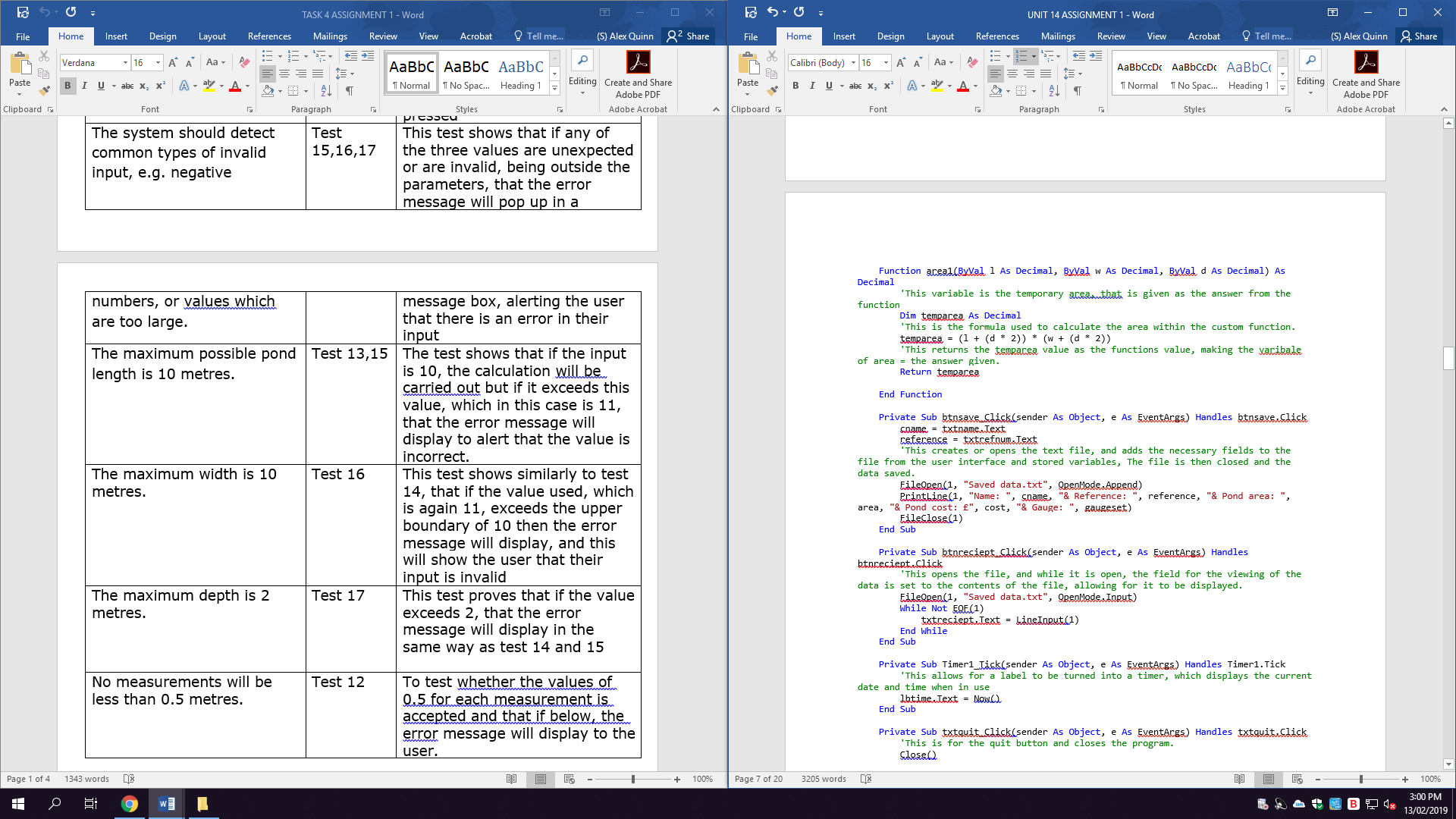
**Have the area, liner and cost calculated in the interface:**

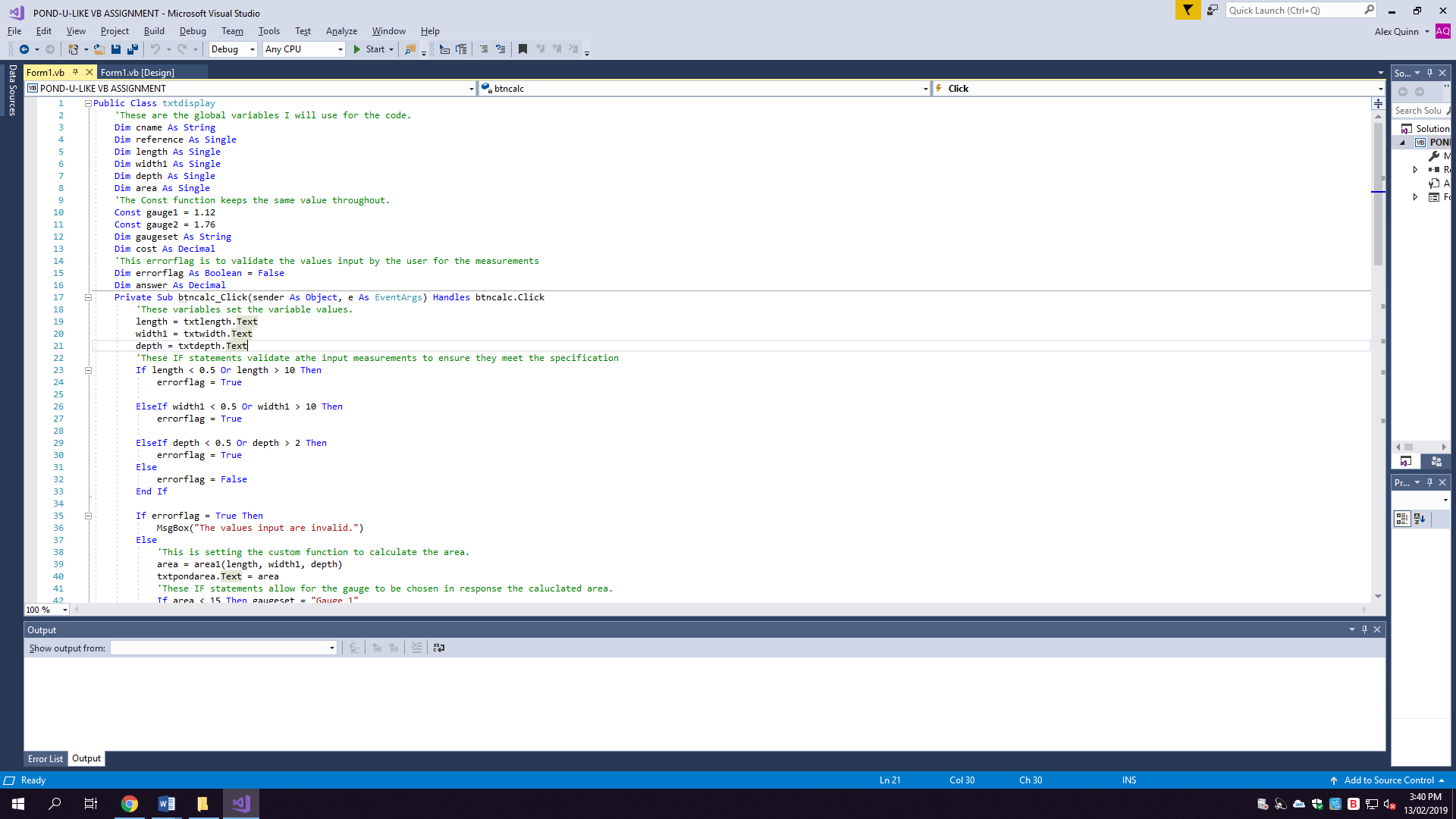
**Using a custom function:**

Once the calculate button is pressed, to meet the specification, the area, gauge and cost are displayed in the corresponding boxes (Tests 6, 7, 8 support this) to display what has been calculated upon the event taking place. This calculate button also consists of a custom function which is required in the specification, this returns the value of the area being “(l + (d \* 2)) \* (w + (d \* 2))” to then display in the textbox. This is then used for the gauge and cost calculation. The function can be used whenever it is needed within the code but is just used in this scenario to generalise the calculation for the values. I have then stored the values calculated into variables that suit the data given. This also gives them a temporary location to be stored both within the program and within the text file once saved and displayed.

As you can see here, the function allows for the values to be passed into “l,w,d” or, length, width and depth. By doing this, it can be called upon multiple times when needed to pass on the value needed without having to repeat the same formula within the code itself.



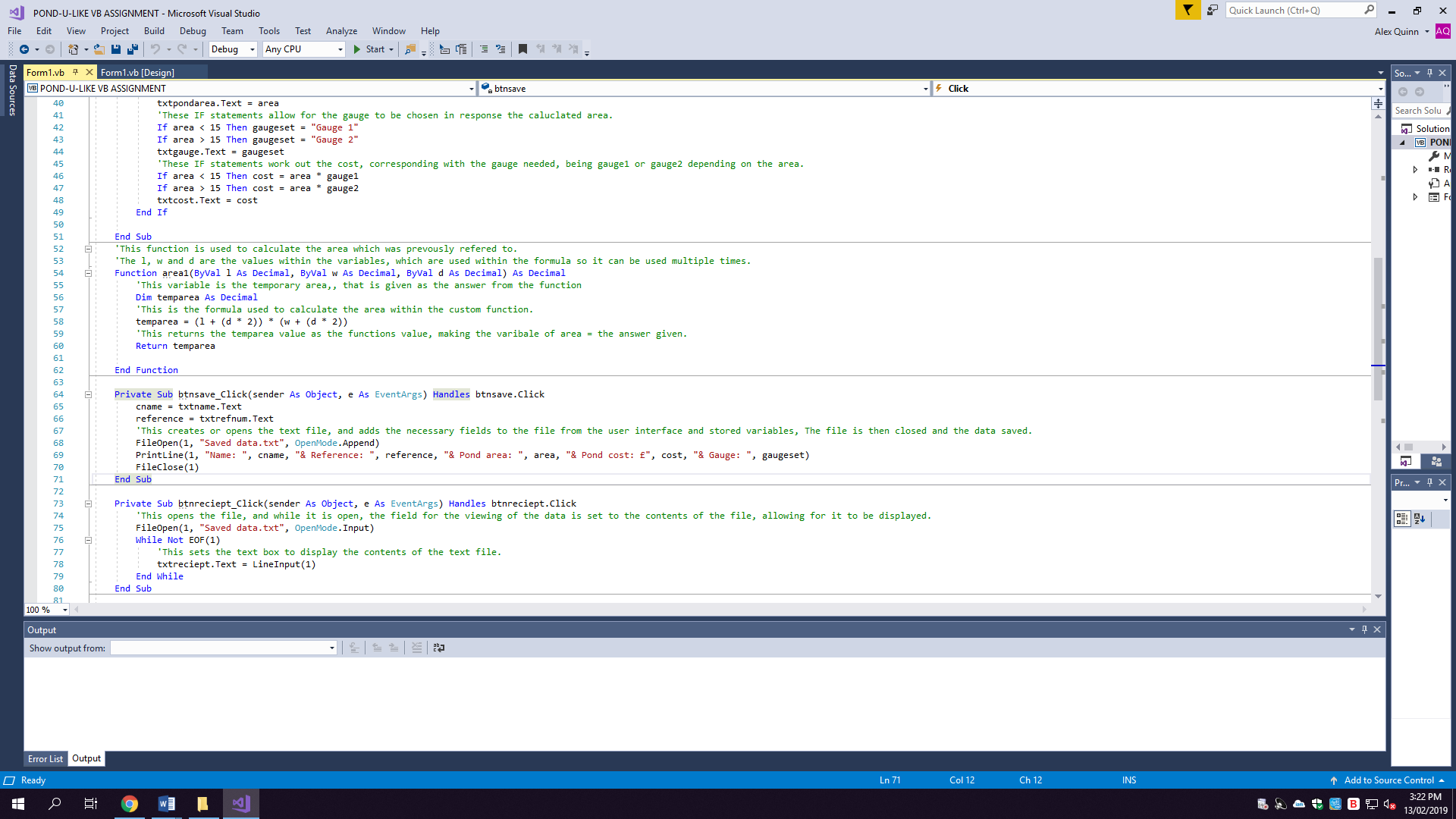




**Be able to save the data into a text file:**

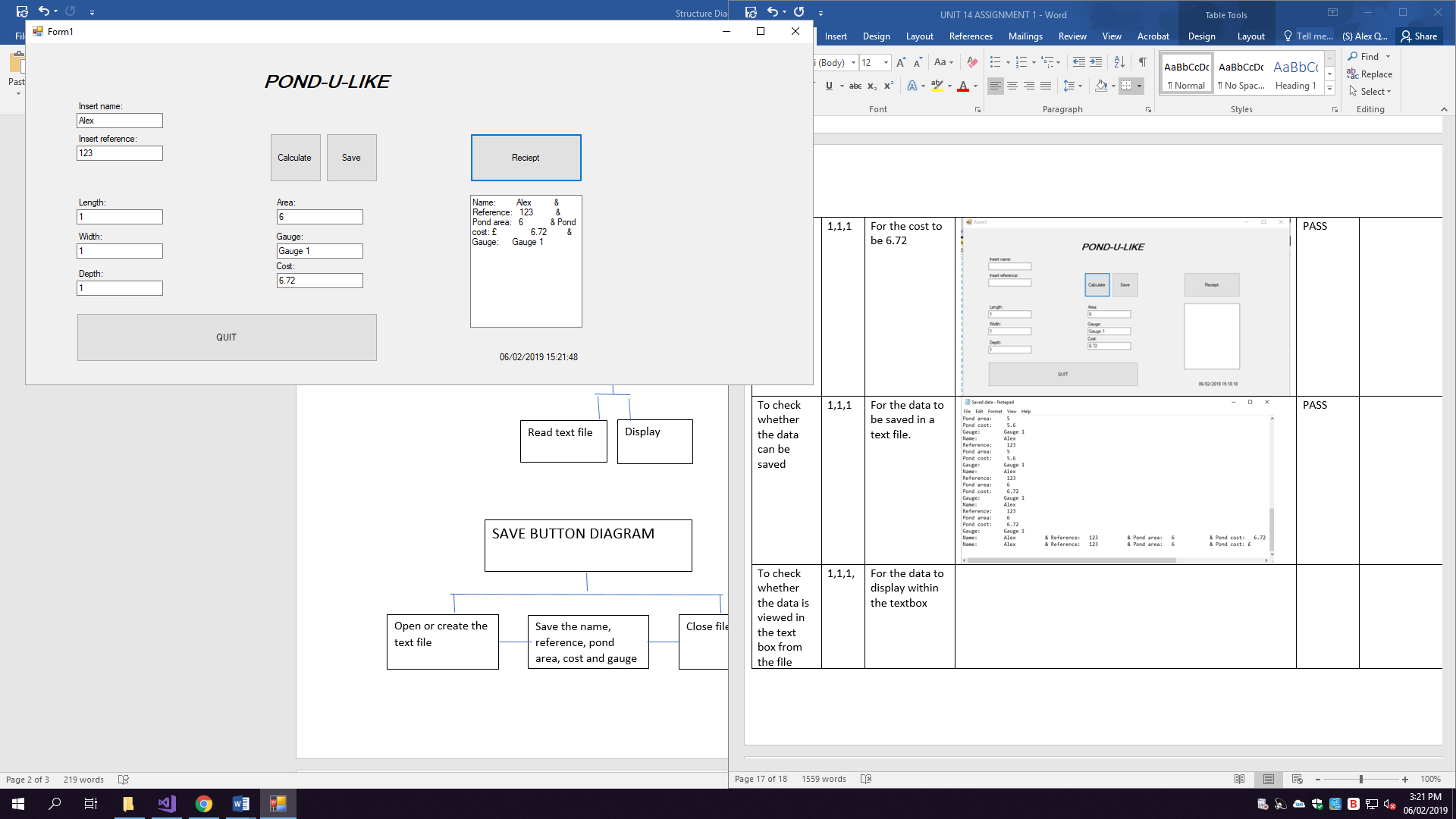
The next step of the program saves the data into a text file, (Test 9 highlights this) which is also specified in the criteria to be later called upon in the next stage of the program. I have done this by creating a relevant text file and saving the data to the same file once, the event of the button being clicked has occurred. The data is saved in the order given in the code. By doing this, it allows for the program to keep a secondary backup of the data while also allowing for it to be displayed in the next step of the program. The text file is saved with a relevant name and has all of the data input from the program, for every use. Allowing a catalogue to be built up.





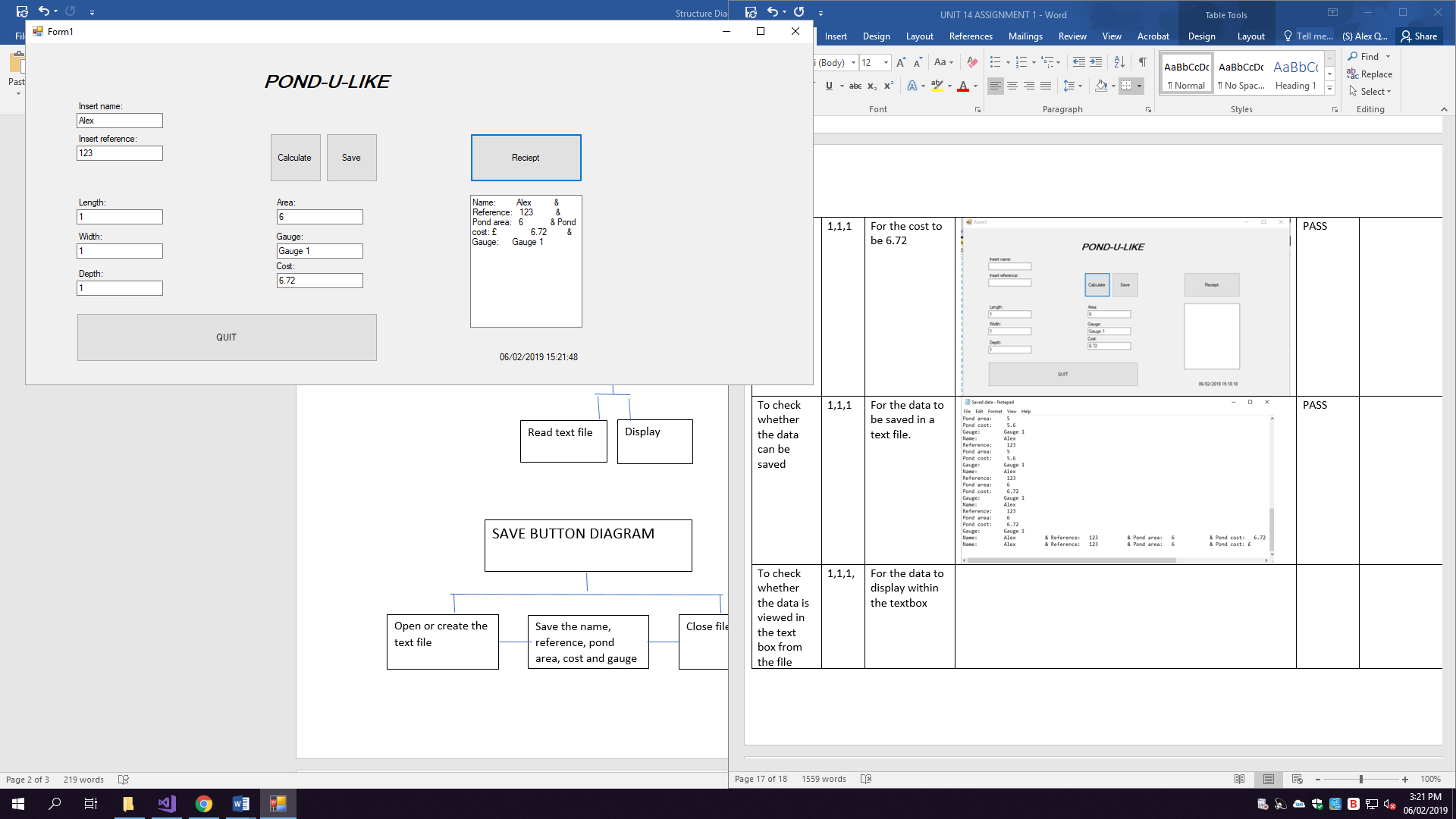
**Be able to retrieve that data in a textbox:**

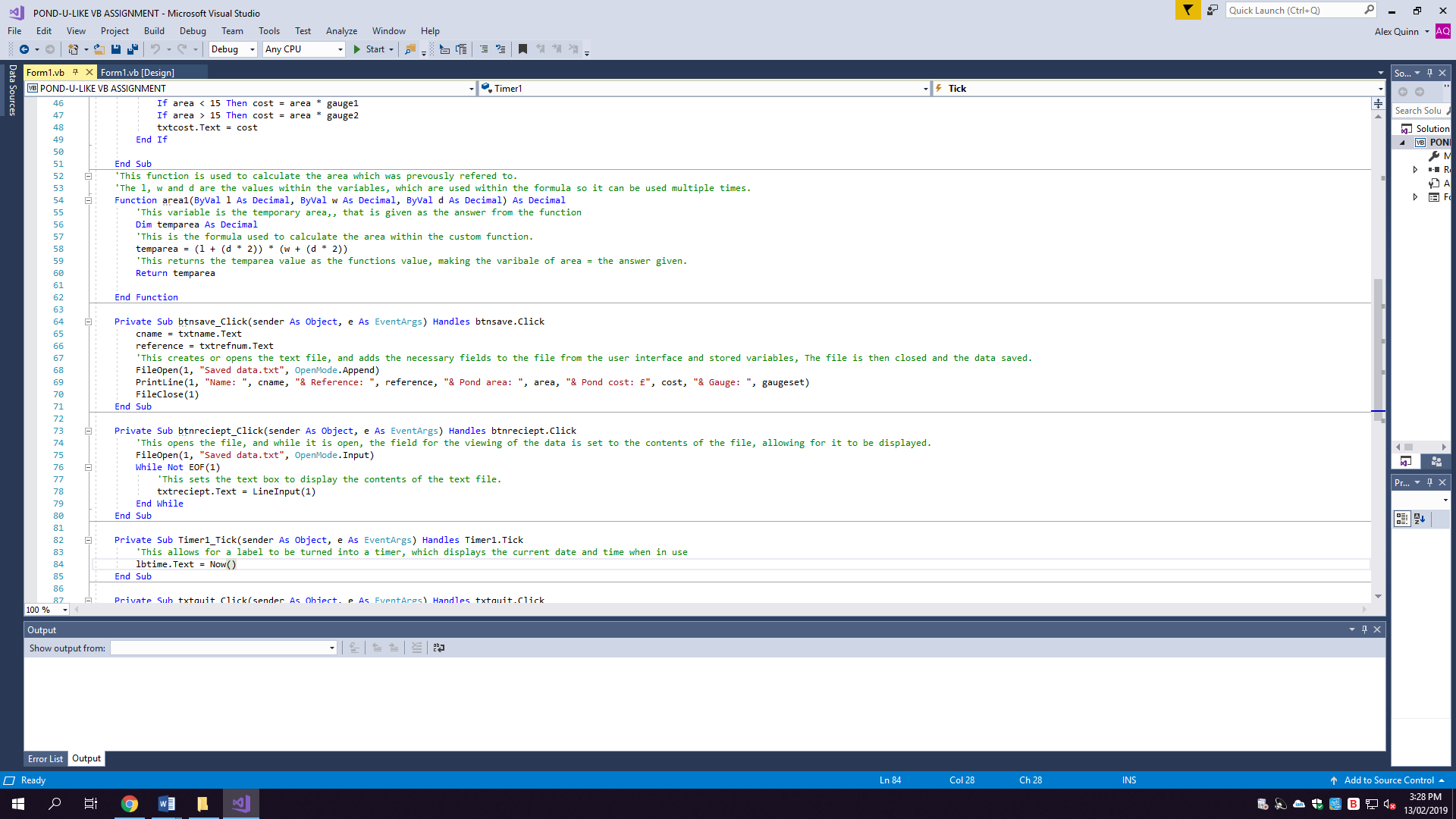
The displaying of the text files content within another textbox (This is highlighted in test 10) which I have labelled as “Receipt”; this is done successfully to meet the main aspects of the specification in relation to the stated requirements. I achieved this through calling upon the previously saved file and setting the textboxes value to the contents of the file. By retrieving the data it meets the criteria and allows it to be viewed by the client or the business. The viewing is only temporary but this is suitable because the box does not need to be filled with every entry ever. This fully meets the criteria of displaying or retrieving the data into a textbox





**Show the current date and time:**

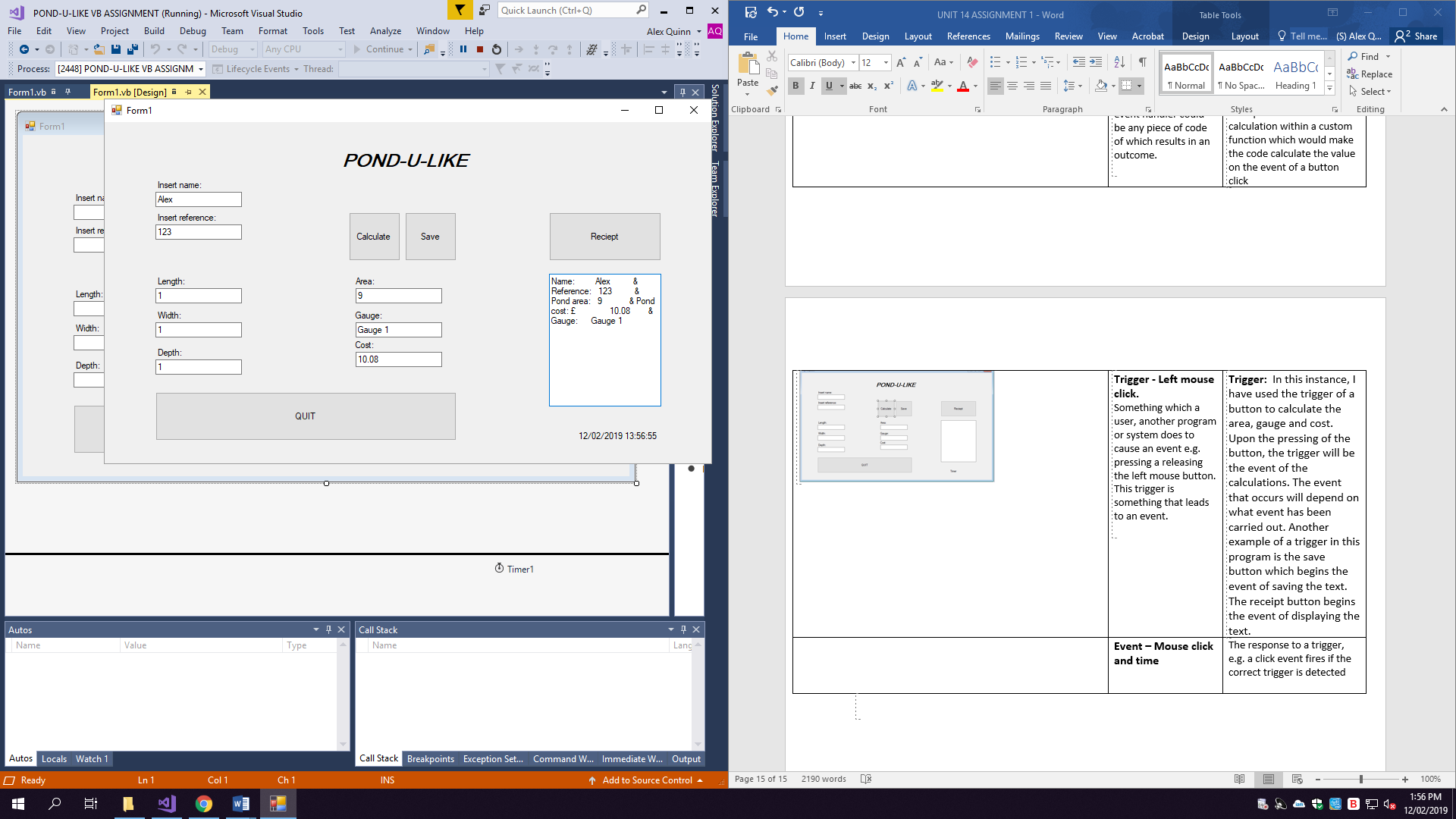
Other aspects of the specification include adding a timer (Highlighted in test 11), which has been done to meet the time driven criteria. I have added the timer at the bottom of the interface so it does not have a prominent influence but can be seen by the user if necessary. I achieved this through setting labels properties to the same name as the timer, while making the timer true to allow it to run.



**Have I tested the code thoroughly?**

I thoroughly tested my program through putting it against extreme value boundaries, by using error flags to validate the figures given. I also tested that the calculations were correct by using a calculator to check the answer. Once the calculations were correct I had tested that all the individual buttons had working routines, while also checking that my custom function successfully corresponded with the area formula, returning a value, which was correct. Once the algorithms were correct I tested the functionality of the save button, which created a text file to store the data into, this was successful. Penultimate, I tested that the viewing button worked in relation to displaying the data within the text box for the viewer to see, in a receipt like format. Lastly, to prove I had fully tested my program I ensured that the quit button closed the program to ensure there was a way to leave or close the program when not in use.

**IMPROVMENTS:**

**VIEW BOX:**

One improvement could be through changing the layout of the text in the view box, in order to make it easier to read. This is because, as seen in the image, that the data is difficult to read as it is not aligned. I could change the arrangement of the variables saved in the file to ensure that they are saved in a list format. Alternatively, I could extend the textbox to allow it all to be saved on one line. In conclusion, changing the arrangement of the data to ensure it is easier to read would improve the ease of use or the ease of viewing when using the program.

**HELP TOOL:**

A help button could be implemented, which could display frequently asked questions when clicked. This would make the program easier to use if the user had any misunderstanding when completing the forms, the misunderstandings could be in regards to the measurement parameters or anything else. With the help button, the interface would have more explanation and the user would potentially have less queries about how it functions. This would be an improvement because it does not currently have this feature, and the creation of a help button would develop the interface in regards to ease of use.

**REFERRENCE SEARCH:**

Adding a tool to make a reference search possible would allow for returning customers to find their data by just searching their number. This would be an improvement because although the data can be saved, specific parts of the data cannot be called up, upon at different times. Meaning this would allow the user to skip the process of completing the form to receive a price etc… With this improvement, figures could be drawn up instantly allowing for a quicker use.

**PRINTING DATA:**

The code could be improved by having a print feature due to the fact that the program and results could then be used as a receipt or as a way to give a customer a quote on a pond liner, this would be an improvement as it would allow for a way to keep physical copies of the client’s data either for Pond-U-Like or for the customer. By having this feature, it would mean that a paper copy of the data could be kept. This would be more practical for use when regarding a quote because the customer could keep it, as opposed to being told a quote and not directly having a means to keep a record of the gauge, price or data.

**The Timer:**

An improvement in relation to the timer could be that the order or quote could be date and time stamped when complete or stored in the text file. This would be an improvement, as it would add accuracy to the form when used in a practical situation.