Workload Distribution

Student Registration No	IT17042352	IT17043588	IT17009096	IT16097520
Student Name	**P.P.G.S.H.A.Guruge**	**M.A.Ashar Ahamed**	**Wellala S.S**	**Ranawake P.I**
Brief description of the function(s) for sprint 1	Implement the functionality required to measure the complexity of a program statement due to type of control structures. This includes identifying conditional and iterative control structures with logical and bitwise operators associated with them. Also detecting the 'catch' and 'switch' statements to calculate the final 'Ctc' value	Measuring the complexity of a program statement due to nesting of control structures. This consists of the functions to outermost nesting levels, immediate inner nesting levels and all the nesting levels beyond that, to come up with the final 'Cnc' value. Calculate the final 'Cp' value based on factor that if the source code contains a recursive function or not	Measuring the complexity of a program statement due to inheritance and come up with the final 'Ci' value. This includes identifying the classes with inheritance and if so the inheritance of all those inherited classes as well. Compute total weight (TW) of a program statement based on previously implemented functions. compute the complexity of a program statement (Cps) based on previously implemented functions. Measuring the complexity introduced due to recursion, to compute the 'Cr' value. This involves detecting the recursive functions correctly.	Implementing the necessary functionality to measure the complexity of a program statement due to size and come up with the final 'Cs' value. This involves detecting the Reference (&) and dereference (*) operators, 'new', 'delete', 'throw', and 'throws' keywords, arithmetic operators, relation operators, logical operators, bitwise operators, miscellaneous operators, assignment operators, keywords, manipulators, text inside a pair of double quotes, class, method, object, variable, and array names and finally numeric values.

Brief description of the function(s) for sprint 2	Generating the final report in PDF format. This includes the complete analysis of source code in a tabular format with all 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values as well as the tokens identified under the size factors.	Function to create the table defined by the size factors and the as well as with a color code. Displaying the 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values in a tabular format in the main UI	Displaying the source code with division in the main UI. These functions would finally display the source code, clearly divided, line by line based on complexity factors with some color code.	Designing and implementing the main UI. This involves design and UX engineering as well as the core placeholder to display the final result in a tabular format. Other team members would display the results in this section. Displaying the tokens identified under the size factors in the relevant column.

Tasks planning to complete in sprint 1 as a developer. (Mention the tasks in point form)	▪ Calculate 'Ctc' value for 'if' conditions ▪ Calculate 'Ctc' value for each logical ('&&' and '||') or bitwise ('&' and '|') operator that is used to combine two or more conditions ▪ Calculate 'Ctc' value for 'for', 'while', or 'do-while' loops ▪ Calculate 'Ctc' value for the 'for', 'while', or 'do-while' loop and for each logical ('&&' and '||') or bitwise ('&' and '|') operator that is used to	▪ Calculate 'Cnc' value for outermost level of nesting of a control structure ▪ Calculate 'Cnc' value for next inner level of nesting of a control structure ▪ Calculate 'Cnc' value for each level of nesting ▪ Calculating the 'Cp' (final complexity) value for a program with one or more recursive methods ▪ Calculating the 'Cp' (final complexity) value for a program without recursive methods	▪ Calculate 'Cci' value for a class due to inheritance ▪ Calculate 'Ci' value for a program statement of a class due to inheritance ▪ Calculate the 'TW' (Total weight) value ▪ Calculate the 'Cps' (complexity of a program statement) value ▪ Calculate 'Cr' value for recursive methods	▪ Calculate 'CS' value for reference (&) and dereference (*) operators ▪ Calculate 'CS' value for 'new', 'delete', 'throw', and 'throws' keywords ▪ Calculate 'CS' value for arithmetic operators ▪ Calculate 'CS' value for relational operators ▪ Calculate 'CS' value for logical operators ▪ Calculate 'CS' value for bitwise operators

	combine two or more conditions ▪ Calculate ‘Ctc’ value for ‘catch’ statements ▪ Calculate ‘Ctc’ value for ‘switch’ statements	▪ Display the final result in a tabular format for spring 1		▪ Calculate ‘CS’ value for miscellaneous operators ▪ Calculate ‘CS’ value for assignment operators ▪ Calculate ‘CS’ value for key words ▪ Calculate ‘CS’ value for text inside a pair of double quotes ▪ Calculate ‘CS’ value for class, method, object, variable, and array names ▪ Calculate ‘CS’ value for numeric values

Tasks planning to complete in sprint 1 as a QA engineer. (Mention the tasks in point form)	▪ Unit testing for 'if' condition detection method. ▪ Unit testing for each logical ('&&' and '||') or bitwise ('&' and '|') operator that is used to combine two or more conditions ▪ Unit testing for iterative control structure detection method ▪ Unit testing for each logical ('&&' and '||') or bitwise ('&' and '|') operator that is used to combine two or more conditions, in an iterative control structure ▪ Testing the 'catch' statement measuring function ▪ Testing the 'switch' statement measuring function	▪ Testing the assignment of weight of zero added for program statements without any level of nesting ▪ Unit testing of the function to add a weight of one for program statements which are at the outermost level of nesting. ▪ Unit testing for the function to add weight of two for program statements which are at the next inner level of nesting ▪ Testing the addition of one to the weight for each level of further nesting. ▪ Testing the detection of source code based on the factor by which if they include recursive methods or not. ▪ Testing the final 'Cp' value calculation for source codes with recursive methods	▪ Testing the calculation of 'Cci' value for a class due to inheritance ▪ Testing the calculation of 'Ci' value for a program statement of a class due to inheritance ▪ Testing the calculation of the 'TW' (Total weight) value ▪ Testing the calculation of 'Cps' (complexity of a program statement) value ▪ Testing the calculation of 'Cr' value for recursive methods	▪ Unit testing for the function to calculate 'CS' value for reference (&) and dereference (*) operators ▪ Unit testing for the function to calculate 'CS' value for 'new', 'delete', 'throw', and 'throws' keywords ▪ Unit testing for the function to calculate 'CS' value for arithmetic operators ▪ Unit testing for the function to calculate 'CS' value for relational operators ▪ Unit testing for the function to calculate 'CS' value for logical operators ▪ Unit testing for the function to calculate 'CS' value for bitwise operators ▪ Unit testing for the function to calculate 'CS' value for miscellaneous operators ▪ Unit testing for the function to calculate 'CS' value for assignment operators ▪ Unit testing for the function to calculate 'CS' value for key words

		▪ Testing the final 'Cp' value calculation for source codes without recursive methods		▪ Unit testing for the function to calculate 'CS' value for text inside a pair of double quotes ▪ Unit testing for the function to calculate 'CS' value for class, method, object, variable, and array names ▪ Unit testing for the function to calculate 'CS' value for numeric values

Tasks planning to complete in sprint 2 as a developer. (Mention the tasks in point form)	▪ Implementing a function to generate a PDF file ▪ Create a table to hold the source code, tokens based on size factors and all 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values, in an A4 size landscape format ▪ Format the rows for source code, tokens and values ▪ create and format columns for each relevant line of source code	▪ Create the basic table to insert the code analysis results ▪ Define the basic color coding for tokens and program statements ▪ Arrange the 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values computed for each line of code ▪ Insert the final results for 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values for each respective line.	▪ Divide the whole table into rows based on the number of lines and properly load the source code into the table ▪ Correctly number the lines ▪ Color code the tokens or lines by the size complexity factors ▪ Properly format the source code line by line in their respective table cell in a way that a line of code with any length can be displayed	▪ Designing the UI ▪ Implementing the UI with basic functions such as loading a file and basic validations ▪ Create a basic placeholder (template) where other team members can refer to, so that they can create a table and load the results which are returned by the functions thy implement. ▪ Display the tokens identified under the size factors in the main table.

	▪ Color code all the relevant lines of code (by token or by line) ▪ Extract data from the existing results in the software and inserting them in the PDF table with correct formatting	▪ Display the final 'Çp' value in the UI table with a proper success or failure message to the user	properly with the available screen size	▪ Create a basic user guide.
Tasks planning to complete in sprint 2 as a QA engineer. (Mention the tasks in point form)	▪ Unit testing for the function to generate a PDF file ▪ Testing the creation of the table to hold the source code, tokens based on size factors and all 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values, in an A4 size landscape format ▪ Testing the formatting of the rows for source code, tokens and values ▪ Testing the creation and formatting ot the columns for each relevant line of source code ▪ Testing the functionality of the color code generation for all the relevant lines of code (by token or by line)	▪ Testing the creation of the basic table to insert the code analysis results ▪ Testing the final color coding for the table ▪ Testing the 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values computed for each line of code ▪ Testing the display of the final results for 'Cs', 'Ctc', 'Cnc', 'Ci', 'TW', 'Cps' and 'Cr' values for each respective line of code. ▪ Unit testing of the function for displaying the final 'Çp' value in the UI table with a proper success or failure message to the user.	▪ Testing the division of the whole table into rows based on the number lines ▪ Unit testing the function to properly load the source code into the table ▪ Unit testing the function to correctly number the lines ▪ Testing the color coding of the tokens or lines by the size complexity factors ▪ Testing the formatting of the source code, line by line in their respective table cells	▪ Prototyping the user interface design ▪ Testing the user experience with the team mates ▪ Testing the UI for basic validation ▪ Unit testing the function to load of the source code file ▪ Testing the main placeholder to load the final table of results ▪ Testing the proper display and representation of the tokens identified under the size factors in the main table.

	▪ Testing the final presentation of the results on the PDF.			▪ Testing the availability and display styling of the user guide