# **Operational Specification Template**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student** | Hristina Koleva | **Date** | 23.10.2013 |
| **Program** | PSP Assignment 7 | **Program #** | 7 |
| **Instructor** | Valentina Ivanova | **Language** | C# |

|  |  |  |  |
| --- | --- | --- | --- |
| Scenario Number | **1** | **User Objective** | Determine the values for - linear regression parameters and correlation coefficients for a set of n pairs,  - improved prediction interval  - prediction intervals:  - the Range for a 70% interval.  - the UPI as P + Range(70%) .  - the LPI as P - Range(70%) .  - significance |
| **Scenario Objective** | User enters a number between 1 and 4 to select test data given data points and data from personal workbook for Program 2 to 6 | | |
| **Source** | **Step** | **Action** | **Comments** |
| **User** | 1 | User launches the calculating application |  |
| Application | 2 | Application is launched |  |
| User | 3 | User enters required input value  User presses Enter then enters the Same value once more and presses Enter once more |  |
| Application | 4 | Application visualizes the result for the input value |  |
|  |  |  |  |
| **Scenario Number** | **2** | **User Objective** | Determine the values for the required parameters (ref. Scenario 1) |
| **Scenario Objective** | *Application validates user input* | | |
| **Source** | **Step** | **Action** | **Comments** |
| **User** | 1 | User launches the calculating application |  |
| Application | 2 | Application is launched |  |
| User | 3 | User enters required input value  User presses Enter |  |
| Application | 4 | Application prompt the user that the calculations cannot proceed with the value entered since it is not a valid one |  |
|  |  |  |  |
|  |  |  |  |

# Functional Specification Template

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Student** | Hristina Koleva | | **Date** | 23.10.2013 | |
| **Program** | PSP Assignment 7 | | **Program #** | 7 | |
| **Instructor** | Valentina Ivanova | | **Language** | C# | |
|  |  | |  | | |
| Class Name | Program | | | | |
| **Parent Class** |  | | | | |
| **Attributes** | | | | | |
|  | **Declaration** | | **Description** | | |
| **Items** | | | | | |
|  | **Declaration** | | **Description** | | |
|  | Main() | | Launches the application and calls the execution of the program | | |
| Class Name | | InputData | | | | |
| **Parent Class** | |  | | | | |
| **Attributes** | | | | | | |
|  | | **Declaration** | | | **Description** | |
|  | | filePath;  fileContent;  x; - first column in the pair for the selected PROBE method  y; - second column in the pair for the selected PROBE method  programFilesDirectory; - used in the user input validation methods    eachLineInProgramFile; - read the files lines by line ; splits the numbers in each row by a comma character and  stores the values of each row in an array then added as the nodes of a linked list.  numbersInARow;  estimatedProxySize; - Constant  listOfRealNumbers; - storing the linked list of the arrays of the values in each column | | | User input to select *the data for calculations:*  *First data set is from file1 containing predefined values for Estimated Proxy Size, Actual and Modified Size and Actual Development times for 10 predefined programs*  *- file2 contains also Estimated Proxy Size, Actual and Modified Size and Actual but from Programs 2 to 6 developed by the student*  *depending on the user input estimated proxy size (386 for data in file 1 and as calculated in the student workbook for file 2) is also set* | |
| **Items** | | | | | | |
|  | | **Declaration** | | | **Description** | |
|  | | ValidateUserInput() | | | Validates the user input from key board as a numeric value | |
|  | | ProcessUserInput() | | | Specifies the data sets/columns from the files to be used in the calculations; specifies the file and the estimated proxy size | |
|  | | CreateInputFile() | | | This is a helper methof used by IsValidInputFile in order to handle missing data file or folder | |
|  | | HandleInvalidUserInput() | | | Provides meaningful user prompt in case of invalid user input | |
|  | | IsValidInputFile() | | | This method validates that the input file exists,  it also validates that the file is not empty and  creates the required input where the user can place the input file. | |
|  | | ReadFile | | | This method reads the values from the input file.  It splits the numbers in each row by a comma character and  stores the values of each row in an array then added as the nodes of a linked list. | |
|  | |  | | |  | |

|  |  |  |
| --- | --- | --- |
| Class Name | RangeValues | |
| **Parent Class** |  | |
|  |  |  |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | dof | Uses *dof* from *UserInput* |
|  | WidthOfI | The range of function parameters |
|  | tInterval | End of widthOfIrange in the current iteration of finding tInterval |
|  | numberOfSegments |  |
|  |  |  |
| **Items** | | |
|  | **Declaration** | **Description** |
|  | GetRangeValues() | widthOfI[i] = i \* tInterval / numberOfSegments; |

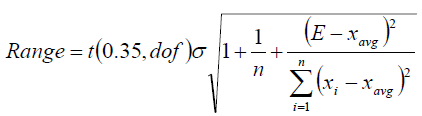
|  |  |  |
| --- | --- | --- |
| Class Name | Terms | |
| **Parent Class** |  | |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | dof | User input to select the degrees of freedom to be used in the calculations of *Student’s*  *t-distribution probability density function* |
|  | functionOfWidth | The range of function parameters |
|  | tInterval | End of *functionOfWidth* range |
|  | SumOfEndTerms | Calculate F(0) + F(X) |
|  |  |  |
| **Items** | | |
|  | **Declaration** |  |
|  | CalculateSumOfTerms() |  |

|  |  |  |
| --- | --- | --- |
| Class Name | Function | |
| **Parent Class** |  | |
|  |  |  |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | dof | User input to select the degrees of freedom to be used in the calculations of *Student’s*  *t-distribution probability density function* |
|  | functionOfWidth | The range of function parameters |
|  | tInterval | End of *functionOfWidth* range |
|  | WidthOfI | X as returned by *RangeValues* |
|  |  |  |
| **Items** | | |
|  | **Declaration** | **Description** |
|  | CalculateGamma() | Calculates the constant function part of gamma function of parameter x = *dof* |
|  | CalculateConstant() |  |
|  | CalculateResult() | Calculates F(X) for each *widthOfI* as m*ultiplies the result of CalculateConstant() and the variable part of the function* and returns the output of the function |

|  |  |  |
| --- | --- | --- |
| Class Name | ProbabilityFunction | |
| **Parent Class** | Program | |
|  |  |  |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | dof | *From UserInput* |
|  | numberOfSegments | *From UserInput* |
|  | Tinterval | *From TValue* |
|  | FunctionOfWidth | Uses result calculated in *CalculateResultFunction()* of *CalculateFunction* class |
|  | probability | The output result of the application, calculated in *CalculateProbability() method* using *firstNumberOfSegments* |
|  | Multiplier | Multiplier = tInterval/numberOfSegments\*3 |
|  |  |  |
| **Items** | | |
|  | **Declaration** | **Description** |
|  | CalculateProbability() | Calculates the output of the application by summarizing the terms and multiplying them by the Width coefficient |

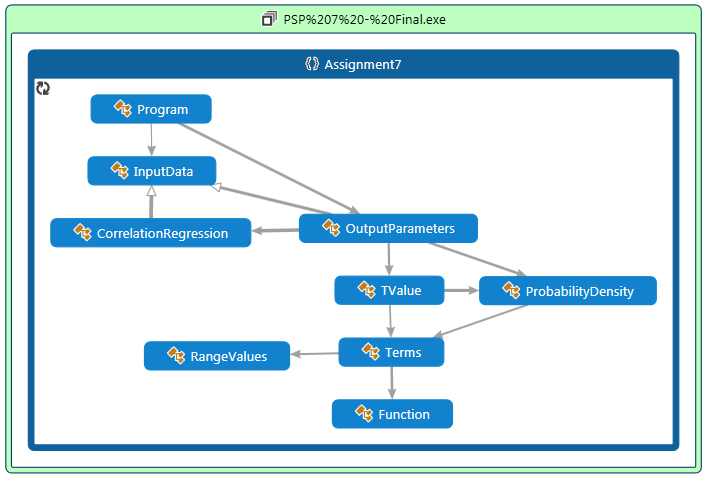
|  |  |  |  |
| --- | --- | --- | --- |
| Class Name | CorrelationRegression | | |
| **Parent Class** | InputData | | |
|  |  |  | | |
| **Attributes** | | | |
|  | **Declaration** | | **Description** |
|  | numberOfSegments; | | *As many as the rows in the file – the data points for each program* |
|  | dof; | | *Degress of freedom for probability density calculation* |
|  | regressionB0; | | The variable storing linear regression parameter |
|  | regressionB1; | | The variable storing linear regression parameter |
|  | improvedPrediction; | | The variable storing the Improved prediction |
|  | correlationR;  correlationR2 | | The variable storing correlation  coefficients *r* and *r* 2 for a set of *n* pairs of data |
|  | meanValueX; | | Helper variables to ease the formulas for regression and correlation parameters |
|  | meanValueY; | |
|  | sumOfXYProducts; | |
|  | sumOfSquareX; | |
|  | productSquareMeanX; | |
|  | productMeanXY; | |
|  | sumOfX; | |
|  | sumOfY; | |
|  | sumOfSquareY; | |
|  | | | |
|  | **Declaration** | |  |
| **Items** | CalculateRegressionParameters | | **regressionB1** = (sumOfXYProducts - productMeanXY) / (sumOfSquareX - productSquareMeanX);      **regressionB0** = meanValueY - (regressionB1 \* meanValueX);    **improvedPrediction** = regressionB0 + regressionB1 \* EstimatedProxySize; |
|  | CalculateCorrelationParameters | | See the formula below  Σ is the symbol for summation  • *i* is an index to the *n* numbers  • *x* and *y* are the two paired sets of data  • *n* is the number of items in each set *x* and *y*  • *avg x* is the average of the *x* values  • *avg y* is the average of the *y* values    r^2 = r\*r |

|  |  |  |
| --- | --- | --- |
| Class Name | OutputParameters | |
| **Parent Class** | InputData | |
|  |  |  |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | significance; | *Variable storing the calculated significance* |
|  | tInterval; |  |
|  | range; | *Variable for the calculated Range* |
|  | sigma; | Variable storing the calculated Standard deviation |
|  | sum; | *Helper variable to ease the formulas for Standard deviation and Range* |
|  | thirdPart; |
|  | sumBelow; |
|  |  |  |
| **Items** | | |
|  | **Declaration** | **Description** |
|  | CalculateSignificance | *1- 2\* p . (The area under the*  curve from –t to t is twice the area from 0 to t, or 2\* *p* ; the remaining area  in the upper and lower tails is 1- 2\* *p* ). |
|  | CalculateNewTForRange | *t*(0.35, *dof*) is the value of *t* for a *t-*distribution for *n* - 2 degrees of freedom  and *p* = 0.35 (program 6) |
|  | CalculateStandardDeviation |  |
|  | CalculateLastPartOfTheProduct |  |
|  | CalculateFinalRange | t\*sigma\*thirdPart  (see formula below) |
|  | CalculateRangeIntervals | Calculate the UPI as *P*  *Range*(70%) .  Calculate the LPI as *P*  *Range*(70%) . |



|  |  |  |
| --- | --- | --- |
| Class Name | TValue | |
| **Parent Class** |  | |
|  |  |  |
| **Attributes** | | |
|  | **Declaration** | **Description** |
|  | tInterval; | *1.0 Trial value initially then the variable stores the calculated t* |
|  | delta; | *0.5* |
|  | errorPrecision; | *0.0000001* |
| **Items** | | |
|  | **Declaration** | **Description** |
|  | FindValueOfT | Find the value of *t* for which integrating Student’s *t*-distribution probability  density function ( *t-distribution pdf*) from 0 to *t* gives a result of *p*.    • Start with a trial value for upper limit of 1 and calculate the value of the  integration.  • Compare it to the desired value.  - if the result of the integration is too low, pick a larger trial upper limit  - if the result of the integration is too high, pick a smaller trial upper limit  The rules for adjusting *d* are these.  1. As long as the tests for the error of the result give the same sign of the error,  leave *d* unchanged.  2. Whenever the sign of the error changes, divide *d* by 2. |

# Logic Specification Template



|  |  |  |  |
| --- | --- | --- | --- |
| Student | Hristina Koleva | **Date** | 23.10.2013 |
| Program | PSP Assignment 7 | **Program #** | 7 |
| Instructor | Valentina Ivanova | **Language** | C# |

|  |  |
| --- | --- |
| **Parameters** | filePath; |
|  | x; |
|  | y; |
|  | programFilesDirectory; |
|  | fileContent; |
|  | eachLineInProgramFile; |
|  | numbersInARow; |
|  | estimatedProxySize; |
|  | listOfRealNumbers; |
|  | dof |
|  | WidthOfI |
|  | tInterval |
|  | numberOfSegments |
|  | functionOfWidth |
|  | SumOfEndTerms |
|  | probability |
|  | Multiplier |
|  | regressionB0; |
|  | regressionB1; |
|  | improvedPrediction; |
|  | correlationR;  correlationR2 |
|  | meanValueX; |
|  | meanValueY; |
|  | sumOfXYProducts; |
|  | sumOfSquareX; |
|  | productSquareMeanX; |
|  | productMeanXY; |
|  | sumOfX; |
|  | sumOfY; |
|  | sumOfSquareY; |
|  | significance; |
|  | range; |
|  | sigma; |
|  | sum; |
|  | thirdPart; |
|  | sumBelow; |
|  | delta; |
|  | errorPrecision; |
|  |  |

|  |
| --- |
| ***ValidateUserInput****()* |
| Read userInput |
| While userInput is valid |
| Call ***ProcessUserInput()*** |
| Else Return |
|  |
| ***ProcessUserInput****()* |
| Case IserInputof |
| UserInput = 1 : filepath = predefined test programs data file; estimatedProxySize = 386; X(ProgramsEstimatedProxySize)= 0; Y(ActualAddedAndModifiedSize) = 2; |
| UserInput = 2: filepath = predefined test programs data file; estimatedProxySize = 386; X(ProgramsEstimatedProxySize)= 0; Y(ActualDevelopmentTime) = 3; |
| UserInput = 3: filepath = file with data from student workbook for Programs 2 to 6; estimatedProxySize = from student workbook; X(ProgramsEstimatedProxySize)= 0; Y(ActualAddedAndModifiedSize) = 1; |
| UserInput = 4: filepath = file with data from student workbook for Programs 2 to 6; estimatedProxySize = from student workbook; X(ProgramsEstimatedProxySize)= 0; Y(ActualDevelopmentTime) = 2; |
|  |
| ***ReadFile()*** |
| ***Read each line in file*** |
| Split by comma |
| Add values in each row to an array |
| Create linked list of corresponding indexes of the array so it contains values from a single column |
|  |
| ***CalculateRegressionParameters()*** |
| NumberOfSegments = LinkedList.Count //number of rows is a column |
| Dof = NumberOfSegments – 2; //as required |
|  |
| regressionB1 = (sumOfXYProducts - productMeanXY) / (sumOfSquareX - productSquareMeanX) |
|  |
| regressionB0 = meanValueY - (regressionB1 \* meanValueX); |
|  |
| improvedPrediction = regressionB0 + regressionB1 \* EstimatedProxySize; |
|  |
| ***CalculateCorrelationParameters()*** |
|  |
| correlationR = (numberOfSegments \* sumOfXYProducts - sumOfX \* sumOfY) / Math.Sqrt((numberOfSegments \* sumOfSquareX - Math.Pow(sumOfX, 2)) \* (numberOfSegments \* sumOfSquareY - Math.Pow(sumOfY, 2))); |
| correlationR2 = Math.Pow(correlationR, 2); |
|  |
| ***CalculateSignificance()*** |
|  |
| tInterval = (Math.Abs(CorrelationR) \* Math.Sqrt(NumberOfSegments - 2)) / (double)Math.Sqrt(1 - CorrelationR2);  CalculateProbability(tInterval, NumberOfSegments, Dof);  significance = 1 - 2 \* Probability; |
|  |
| ***CalculateNewTForRange()*** |
|  |
| Probability = 0.35; |
| Call FindValueOfT() with this Probability |
|  |
| ***CalculateStandardDeviation()*** |
| For each row in selected column from file |
| Sum += Math.Pow((Yi – regressionb0 - regressionb1\*Xi),2) |
| sigma = Math.Sqrt((1 / NumberOfSegments - 2 \* sum); |
|  |
| ***CalculateLastPartOfTheProduct()*** |
| For each row in selected column from file |
| SumBelow += Math.Pow((Xi – MeanX),2) |
|  |
| ThirdPart = Math.Sqrt(1+1/numberOfSegments + Math.Pow(estimatedProxySize – MeanX),2)/sumBelow) |
|  |
| ***CalculateFinalRange()*** |
|  |
| Range = tInterval\*Sigma\*thirdPart |
|  |
| ***CalculateRangeIntervals()*** |
|  |
| LPI = ImprovedPrediction - Range |
| UPI = ImprovedPrediction + Range |
|  |
|  |
| ***FindValueOfT()*** |
| Tinterval = 1.0 |
| Delta = 0.5 |
| ErrorPrecision = 0.0000001 |
| Call ***CalculateProbability()*** *return Probability* |
|  |
| ***While (NOT !(probability – probability <= E))*** |
|  |
| **IF**(Math.Sign(probability – Probability) <> Math.Sign(errorPrecision) |
| { |
| Delta = delta/2 |
| If (probability > Probability) |
| { |
| Tinterval = tInterval + delta |
| Call ***CalculateProbability()*** |
| } |
| Else if (probability < Probability) |
| { |
| Tinterval = tInterval - delta |
| Call ***CalculateProbability()*** |
| } |
| } |
| Else if (Math.Sign(probability – Probability) == Math.Sign(errorPrecision) |
| { |
| If (probability > Probability) |
| { |
| Tinterval = tInterval + delta |
| Call ***CalculateProbability()*** |
| } |
| Else if (probability < Probability) |
| { |
| Tinterval = tInterval - delta |
| Call ***CalculateProbability()*** |
| } |
| } |
| Return; |
| ***GetRangeValues****()* |
| For I = 0; I <numberOfsegments+1; I ++ |
| WidthOfI[i] = i\*tInterval/numberOfSegments |
|  |
| ***CalculateGamma****(dof)* |
| If dof == 1 return 1 |
| If dof == 0.5 return |
| Return (dof-1)\*CalculateGamma(dof+1) |
|  |
| ***CalculateConstant****()* |
| Return CalculateGamma((dof + 1) / 2) /\*CalculateGamma(dof/2) |
|  |
| **CalculateResult**() |
| For I = 0; I < numberOfSegments +1; i++ |
| FunctionOfWidth[i] = CalculateConstant(dof)\* |
|  |
| ***CalculateSumOfTerms****()* |
| SumOfEndTerms = funtionOfWidth[0] + functionOfWidth[tInterval] |
|  |
| For I = 1; I < numberOfSegments; I+=2 |
| sumOfOddTerms[i] = functionOfWidth[i]\*4 |
| SumOfTerms += sumOfOddTerms[i] |
|  |
| For J = 2; J < numberOfSegments; J +=2 |
| sumOfEvenTerms[j] = functionOfWidth[i]\*2; |
|  |
| SumOfTerms += sumOfEndTerms; |
|  |
| ***CalculateProbability()*** |
|  |
| Call ***CalculateSumOfTerms()*** |
|  |
| Multiplier = tInterval/numberOfSegments\*3 |
|  |
| Probability = multiplier\*sumOfTerms |
|  |
| Console.WriteLine(TInterval = {0:F5}, tInterval); |