Code Review Check List
For C# Language

Project ID:		Work product:	
Checked By:		Date :	05 de Fevereiro de 2023
Note:			

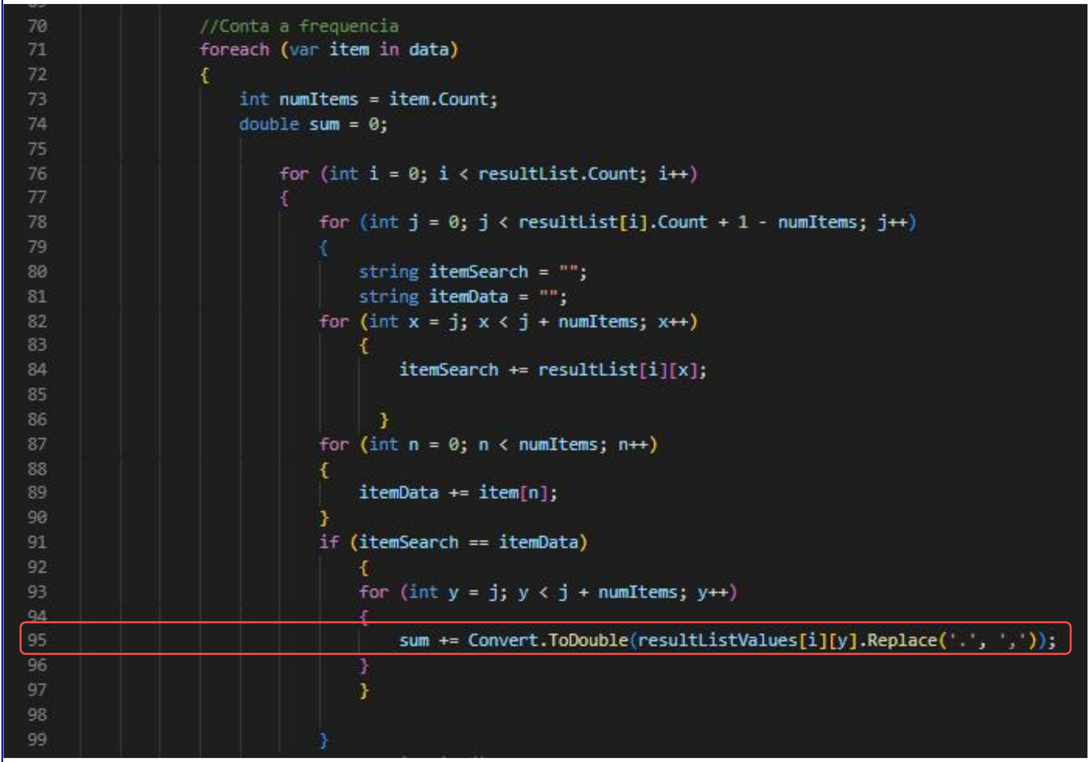
I - DEVIATION OBJECTIVE				
#	I.1 – DEVIATION	**Yes**	**No**	**NA**
1.	Does the code correctly implement the design?	☒	☐	☐
2.	Does the code implement more than the design?	☐	☒	☐
3.	Is every parameter of every method passing mechanism (value or reference) appropriate?	☒	☐	☐
4.	Does every method return the correct value at every method return point?	☒	☐	☐
II – OMISSION OBJECTIVE				
#	II.1 –OMISSION	**Yes**	**No**	**NA**
5.	Does the code completely implement the design?	☒	☐	☐
6.	Are there any requirements of design that were not implemented?	☐	☒	☐
III - DEFECT OBJECTIVE				
#	III.1 – Variable and Constant Declaration	**Yes**	**No**	**NA**
7.	Are descriptive variable and constant names used in accord with naming conventions?	☒	☐	☐
8.	Is every variable correctly typed?	☒	☐	☐
9.	Is every variable properly initialized?	☒	☐	☐
10.	Are all for-loop control variables declared in the loop header?	☒	☐	☐
11.	Are there variables that should be constants?	☐	☒	☐
12.	Are there attributes that should be local variables?	☐	☒	☐
13.	Do all attributes have appropriate access modifiers (private, protected, public)?	☒	☐	☐
14.	Are there static attributes that should be non-static or vice-versa?	☐	☐	☒
#	III.2 – Method Definition	**Yes**	**No**	**NA**
15.	Are descriptive method names used in accord with naming conventions?	☒	☐	☐
16.	Do all methods have appropriate access modifiers (private, protected, public)?	☒	☐	☐
17.	Is every method parameter value checked before being used?	☒	☐	☐
18.	Are there static methods that should be non-static or vice-versa?	☐	☐	☒
#	III.3 – Class Definition	**Yes**	**No**	**NA**
19.	Does each class have an appropriate constructor?	☒	☐	☐
20.	Do any subclasses have common members that should be in the superclass?	☐	☒	☐
21.	Can the class inheritance hierarchy be simplified?	☐	☒	☐
#	III.4 – Data Reference	**Yes**	**No**	**NA**
22.	For every array reference: Is each subscript value within the defined bounds?	☒	☐	☐
23.	For every object or array reference: Is the value certain to be non-null?	☒	☐	☐
#	III.5 – Computation/Numeric	**Yes**	**No**	**NA**
24.	Are there any computations with mixed data types?	☐	☒	☐
25.	Is overflow or underflow possible during a computation?	☒	☐	☐
26.	For each expression with more than one operator: Are the assumptions about order of evaluation and precedence correct?	☒	☐	☐
27.	Are parentheses used to avoid ambiguity?	☒	☐	☐
28.	Does the code systematically prevent rounding errors?	☒	☐	☐
29.	Does the code avoid additions and subtractions on numbers with greatly different magnitudes?	☒	☐	☐
30.	Are divisors tested for zero or noise?	☒	☐	☐

#	III.6 – Comparison/Relational	Yes	No	NA
31.	Has each boolean expression been simplified by driving negations inward?	☒	□	□
32.	For every boolean test: Is the correct condition checked?	☒	□	□
33.	Are there any comparisons between variables of inconsistent types?	□	☒	□
34.	Are the comparison operators correct?	☒	□	□
35.	Is each boolean expression, correct?	☒	□	□
36.	Are there improper and unnoticed side-effects of a comparison?	□	☒	□
37.	Has an "&" inadvertently been interchanged with a "&&" or a "|" for a "||"?	□	☒	□
38.	Does the code avoid comparing floating-point numbers for equality?	☒	□	□
39.	Is every three-way branch (less,equal,greater) covered?	☒	□	□
#	III.7 – Control Flow	Yes	No	NA
40.	For each loop: Is the best choice of looping constructs used?	☒	□	□
41.	Will all loops terminate?	☒	□	□
42.	When there are multiple exits from a loop, is each exit necessary and handled properly?	☒	□	□
43.	Does each switch statement have a default case?	☒	□	□
44.	Are missing switch case break statements correct and marked with a comment?	□	□	☒
45.	Is the nesting of loops and branches too deep, and is it correct?	□	□	☒
46.	Can any nested if statements be converted into a switch statement?	□	☒	□
47.	Are null bodied control structures correct and marked with braces or comments?	☒	□	□
48.	Does every method terminate?	☒	□	□
49.	Are all exceptions handled appropriately?	☒	□	□
50.	Do named break statements send control to the right place?	☒	□	□
#	III.8 – Input/Output	Yes	No	NA
51.	Have all files been opened before use?	☒	□	□
52.	Are the attributes of the open statement consistent with the use of the file?	☒	□	□
53.	Have all files been closed after use?	☒	□	□
54.	Is buffered data flushed?	☒	□	□
55.	Are there spelling or grammatical errors in any text printed or displayed?	□	☒	□
56.	Are error conditions checked?	☒	□	□
57.	Are files checked for existence before attempting to access them?	☒	□	□
58.	Are all I/O exceptions handled in a reasonable way?	☒	□	□
#	III.9 – Module Interface	Yes	No	NA
59.	Are the number, order, types, and values of parameters in every method call in agreement with the called method's declaration?	☒	□	□
60.	Do the values in units agree (e.g., inches versus yards)?	□	□	☒
61.	If an object or array is passed, does it get changed, and changed correctly by the called method?	☒	□	□
#	III.10 – Comment	Yes	No	NA
62.	Does every method, class, and file have an appropriate header comment?	□	☒	□
63.	Does every attribute, variable or constant declaration have a comment?	□	☒	□
64.	Is the underlying behavior of each method and class expressed in plain language?	□	☒	□
65.	Is the header comment for each method and class consistent with the behavior of the method or class?	□	☒	□
66.	Are all comments consistent with the code?	□	☒	□
67.	Do the comments help in understanding the code?	☒	□	□
68.	Are there enough comments in the code?	□	☒	□
69.	Are there too many comments in the code?	□	☒	□
#	III.11 – Layout and Packing	Yes	No	NA
70.	Is a standard indentation and layout format used consistently?	☒	□	□
71.	For each method: Is it no more than about 60 lines long?	□	☒	□

72.	For each compile module: Is no more than about 600 lines long?	☒	□	□
#	III.12 – Modularity	Yes	No	NA
73.	Is there a low level of coupling between modules (methods and classes)?	□	☒	□
74.	Is there a high level of cohesion within each module (methods or class)?	□	☒	□
75.	Is there repetitive code that could be replaced by a call to a method that provides the behavior of the repetitive code?	□	☒	□
76.	Are the Java class libraries used where and when appropriate?	□	□	☒
#	III.13 – Storage Usage	Yes	No	NA
77.	Are arrays large enough?	☒	□	□
78.	Are object and array references set to null once the object or array is no longer needed?	□	☒	□
#	III.14 – Performance	Yes	No	NA
79.	Can better data structures or more efficient algorithms be used?	□	☒	□
80.	Are logical tests arranged such that the often successful and inexpensive tests precede the more pensive and less frequently successful tests?	□	□	☒
81.	Can the cost of recomputing a value be reduced by computing it once and storing the results?	□	☒	□
82.	Is every result that is computed and stored actually used?	☒	□	□
83.	Can a computation be moved outside a loop?	□	☒	□
84.	Are there tests within a loop that do not need to be done?	□	☒	□
85.	Can a short loop be unrolled?	□	☒	□
86.	Are there two loops operating on the same data that can be combined into one?	□	☒	□
87.	Are frequently used variables declared register?	☒	□	□
88.	Are short and commonly called methods declared inline?	☒	□	□
89.	Are timeouts or error traps used for external device accesses?	□	□	☒
IV - INCONSISTENCY OBJECTIVE				
#	IV.1 – Performance	Yes	No	NA
90.	Are there any code implement in inconsistent way?	□	☒	□
V – AMBIGUITY OBJECTIVE				
#	V.1 – Variable and Constant Declaration	Yes	No	NA
91.	Are there variables with confusingly similar names?	□	☒	□
92.	Are all variables properly defined with meaningful, consistent, and clear names?	☒	□	□
#	V.2 – Performance	Yes	No	NA
93.	Are any modules excessively complex and should be restructured or split into multiple routines?	□	☒	□
VI – REDUNDANCE OBJECTIVE				
#	VI.1 – Variables	Yes	No	NA
94.	Are there any redundant or unused variables or attributes?	□	☒	□
95.	Could any non-local variables be made local?	□	☒	□
#	VI.2 – Method Definition	Yes	No	NA
96.	Are there any uncalled or unneeded methods?	□	☒	□
#	VI.3 – Performance	Yes	No	NA
97.	Can any code be replaced by calls to external reusable objects?	□	☒	□
98.	Are there any blocks of repeated code that could be condensed into a single method?	□	☒	□
99.	Are there any leftover stubs or test routines in the code?	□	☒	□
VII – SIDE-EFFECT OBJECTIVE				
#	VII.1 – Method Definition	Yes	No	NA
100.	After changing of prototype of method, Have class which calls it considered yet?	□	□	☒
#	VII.2 – Data Base	Yes	No	NA
101.	Do Upgrading and Migration process follow up changing of structures or contents of a project’s data base?	□	□	☒

VIII – Paths of improvements

#	VIII.1 – Major identified errors
102.	Se nas definições regionais do Windows, tiver no separador decimal uma vírgula em vez do ponto, o programa encerra.



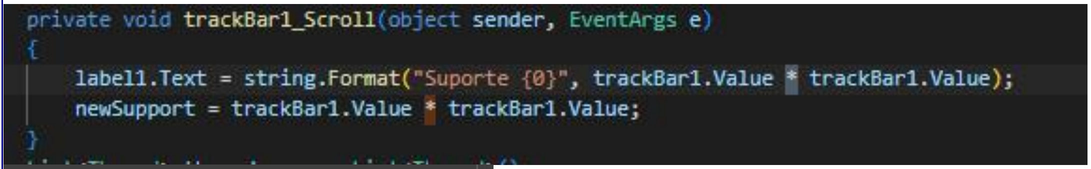
```
//Conta a frequencia
foreach (var item in data)
{
    int numItems = item.Count;
    double sum = 0;

    for (int i = 0; i < resultList.Count; i++)
    {
        for (int j = 0; j < resultList[i].Count + 1 - numItems; j++)
        {
            string itemSearch = "";
            string itemData = "";
        for (int x = j; x < j + numItems; x++)
        {
            itemSearch += resultList[i][x];

          }
        for (int n = 0; n < numItems; n++)
        {
            itemData += item[n];
        }
        if (itemSearch == itemData)
        {
            for (int y = j; y < j + numItems; y++)
            {
                sum += Convert.ToDouble(resultListValues[i][y].Replace('.', ','));
            }
            }

        }
```

#	VIII.2 – Improvements suggestions
103.	O nível do suporte mínimo do algoritmo não era o mais adequado quando se aplicavam os filtros (situação já corrigida).



```
private void trackBar1_Scroll(object sender, EventArgs e)
{
    label1.Text = string.Format("Suporte {0}", trackBar1.Value * trackBar1.Value);
    newSupport = trackBar1.Value * trackBar1.Value;
}
```