

Lab Practicum #2

Using Code Metrics for software analysis

Background:

Developers need to have feedback about the quality of code immediately after the code is written. Team Edition for Software Developers which is part of Visual Studio Team System provides an integrated tool named Code Metrics to calculate how the code is written based upon some quantifiable criterion and shows the results in a separate window. Calculations are based upon criterions like *Cyclomatic complexity*, *number of lines in code and inheritance depth* etc, all resulting in maintainability index of the code.

Code Metrics provides feedback to developers on following factors:

1. Lines of Code

Measuring programming progress by lines of code is like measuring aircraft building progress by weight. (Bill Gates)

This one is simple one. This metric is most famous for its misuse. So use it carefully, ideally for information only, to see how large the class or method is.

2. Maintainability Index: this index is a computed using a formula as illustrated in [Visual Studio Code Analysis Team Blog](#): $\text{Maintainability Index} = 171 - 5.2 * \log_2(\text{Halstead Volume}) - 0.23 * (\text{Cyclomatic Complexity}) - 16.2 * \log_2(\text{Lines of Code})$. This formula is based upon the work done at Carnegie Melon University.

Range of values for maintainability index is from 0 to 100 where 0 is the least maintainable code and 100 is perfectly maintainable code. Naturally higher values i.e. the ones tending towards 100 are going to be welcome. To make the things simple, code metrics tool in its results provides icons for the part of the code (a class or a method) along with index values to show how much maintainable it is. Red icon represents maintainability index values less than 10 which mean that those parts of the code need immediate attention of the developer. Values between 10 and 20 are accompanied with yellow icon to bring to notice the parts of the code which are not well maintainable. All the values above 20 are adorned with green icon to show that the represented part of the code is maintainable.

3. Cyclomatic Complexity: This value is the count, for each part of the code, of various branches in the part of the code. Branches can be because of various statements which can make decisions like if and switch statements or they can be loop statements like for, while, do etc. Lower values are better as they provide more readable and maintainable code.

4. Class Coupling: This value is a count for each class on which that class depends. If the coupling count for a class is high which means that it depends upon many other classes then the chances that the code of this class will break due to some changes in the dependency classes is high. That is why as a good practice, this

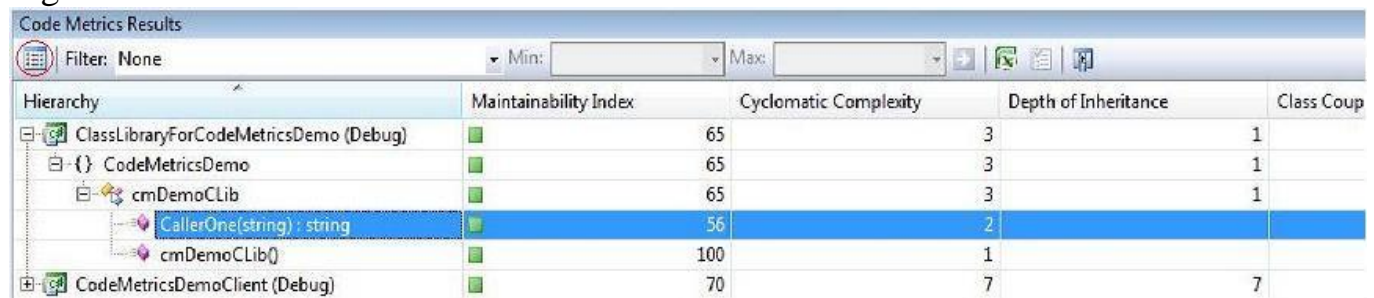
number should be as low as possible. Solution is to provide loose coupling or Service Oriented Architecture if possible. To identify common components and abstract them as a service will be a good practice. Class coupling number does not take into account the primitives like int32 and String etc.

5. Inheritance Depth: This number represents the number of types which are above the type being reported in the inheritance tree. This value is measured from Object class which is at level 0.

Results Window for Code Metrics

Results of the code metrics calculation are made available in a separate window which can be opened from either the View menu or from the Analyze menu. Once the window is opened we can calculate the code metrics by clicking on the analyze button as shown with a red circle in Figure 1. Results are shown in a grid which has a row for each module (e.g. class, method) for which code metrics can be computed. For each measurable criterion as are mentioned above, there is a column in the grid. Figure 1 shows the Results Window

Figure 1 shows the Results Window



Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coup
ClassLibraryForCodeMetricsDemo (Debug)	65	3	3	1
CodeMetricsDemo	65	3	3	1
cmDemoCLib	65	3	3	1
CallerOne(string) : string	56	2	2	
cmDemoCLib()	100	1	1	
CodeMetricsDemoClient (Debug)	70	7	7	7

Figure 1

There is a possibility to filter the results of code metrics based upon values of any of the columns in the results window. To set the filter we can use the menu bar of the code metrics results window. In the menu bar you can select the column which will be the criterion and then set the minimum and maximum values which we want to set filter on. It is not possible to set the complex filter.

It is also possible to remove some of the columns or add them back again in the results window. It is also possible to change the sequence in which they appear from left to right.

You are able to export the results of code metrics either to the clipboard or to MS Excel in a table form. These results then can be used to do further offline analysis.

One of the best features for management is the ability to create work items directly from the code metrics results window. We can use the context menu to create various types of work items and assign them to developers for refactoring the code which does not have good maintainability.

Limitations and Constraints

Some of the things that code metrics feature of Team Edition for Software Developers cannot do are as follows:

1. Calculating code metrics for only part of the code in a module.
2. Providing custom columns in the code metrics results window.

3. Pinpointing the code lines due to which maintainability of the code of a method is reduced.
4. Exclusion of certain type of code like generated code from metrics calculations and results.
5. Calculation of code metrics cannot be set as check-in policy

Goal of the lab:

The objective of this lab is to learn how to use the code metrics tools within VSTS for analyzing code.

Task description:

- 1) Download and install RSS Bandit application and its source code (see Input Materials section).
- 2) Open source code in Visual Studio 2008 .
- 3) Generate Code Metrics for the entire solution;
- 4) Analyze results; find out where the “rotten code” is located and try to make refactoring;
- 5) Create report with generated code metrics table and with improved code;
- 6) Learn code metrics theory (See background) for the lab’s defence.

Input Materials:

- 1) RSS Bandit application <http://rssbandit.org/>
 - a. <http://sourceforge.net/projects/rssbandit/files/rssbandit/v1.8.0.870/RssBandit.1.8.0.862.Src.zip/download>
 - b. http://sourceforge.net/projects/rssbandit/files/rssbandit/v1.8.0.870/RssBandit1.8.0.870_installer.zip/download