

Frestimate Metrics User's Manual

Version 1.3

SoftRel, LLC

www.softrel.com







Frestimate Metrics User's Manual	1
Overview of Frestimate Metrics	2
Number of executable lines of code	3
Number of commented lines.....	3
Number of inline comments	3
Percentage of comments	3
Blank lines	3
Total lines of code	3
Cyclomatic complexity	3
Cyclomatic complexity due to case statements	4
Cyclomatic complexity due to exception handling	4
Number of GOTO statements	4
Depth of nesting.....	4

Overview of Frestimate Metrics






The Frestimate Metrics package is designed to compute several of the inputs required for the Frestimate prediction models. Ideally, you need to predict the size BEFORE the code is written. One way to do this is to measure the size from the previous version, establish it as a baseline and then predict the size of the next version as a percentage change of the last version. Some organizations even plot the actual size of each software version and then develop a model to predict size based on certain characteristics such as people and calendar time. The first step, however, is establishing the actual size of a past version of software. This is where Frestimate Metrics can help.

The metrics computed include:

Metrics for estimating size

-  Number of executable lines of code
-  Number of commented lines of code
-  Number of inline
-  Percentage of comments
-  Blank lines
-  Total lines of code

Other metrics

-  Cyclomatic complexity
-  Cyclomatic complexity attributed to case statements
-  Cyclomatic complexity attributed to exception handling
-  Depth of nesting
-  Number of GOTO statements

General instructions:

The above metrics are language specific. You can purchase one or more of the below language tools. When you run FrestimateMetrics you can choose any of the languages that you have purchased for the metrics report.



- Ada
- VB
- VB .net
- C++
- Java

You will need to select the location where the source files are located. This can be any location that your computer has access to. Once you have selected the location of the source files, you can select whether or not to include up to 3 levels of subdirectories in the complexity report. If you don't select the subdirectories then only those source files in the selected directory will be measured.

The FrestimateMetrics program should be installed in the same folder as the Frestimate software if you wish to launch the metrics modules from Frestimate. You can run FrestimateMetrics from another folder; however, you won't be able to import your results directly into Frestimate.

Number of executable lines of code

This includes all lines of code that are **not**:

-  Fully commented
-  Blank

Lines of code that contain inline comments are counted towards executable lines and towards the inline comment count but not towards the commented line count. These are accumulated for each function as well as each file. This metric is also called SLOC for Source Lines Of Code. When SLOC is divided by 1000 it is called KSLOC for 1000 SLOC. KSLOC is input to the Frestimate General Inputs page and is used by all of the prediction models.

Number of commented lines

This includes all lines of code that are completely commented and therefore completely ignored by the compiler/preprocessor. Inline comments are counted separately. These are accumulated for each function as well as each file. The commented lines should not be included in a size prediction as only EFFECTIVE lines of code are counted in Frestimate.

Number of inline comments

An inline comment is on the same line as an executable line of code. These are counted separately from the commented lines. The inline comment line will also be counted as an executable line. These are accumulated for each function as well as each file.

Percentage of comments

This is the number of fully commented lines of code divided by the total number of lines of code. The percentage of comments is used as an indicator of readability.

Blank lines

This is the number of lines that have nothing on them at all. This is counted at the function and file level. Blank lines are not included in size predictions used in Frestimate.

Total lines of code

This is the sum of the executable lines and the commented lines of code. Blank lines are not counted. This number is used mostly to compute the percentage of comments.

Cyclomatic complexity

This is equal to the number of branches in logic plus one. It is computed at the function level and then accumulated at the file level. The cyclomatic complexity due to case statements or exception handling is reported separately as shown below.

Examples of branches in logic are:

If- then-else-elseif

Loops (For, Repeat until and while)

Cyclomatic complexity due to case statements

A case statement is a branch in logic that has several possible branches which are all independent and not sequential in processing order. The complexity due to the case statement is simply the sum of all cases within the case statement. The complexity due to case statements is often reported separately because it is often unavoidable.

Cyclomatic complexity due to exception handling

Exception handling is necessary for ensuring the reliability and robustness of the software. Unfortunately, exception handling also adds to the complexity of the software. Since exception handling is a necessary complexity, it is computed separately. The branches in logic which are known to be exception handling include:

Try, Catch, Finally (C++ and Visual Basic .Net)
On error GOTO (VB)
Exception (Ada)

Note that for VB, the On Error GOTO is not counted towards the number of GOTO statements since it is counted towards exception handling.

Number of GOTO statements

A GOTO statement is an unstructured branch in logic. The GOTO statement is the same in every language. For Visual Basic the "On Error GOTO" is counted towards the exception handling count but not towards the GOTO count. The number of GOTO statements is an input for the Rome Laboratory prediction model.

Depth of nesting

The depth of nesting is a measure of how many sequential layers of logic there are. This is an example of a depth of nesting that is equal to 3. Theoretically, the more layers the less readable the software is. However, the only feasible way to reduce layers is to have complex logic, which can be dangerous since the complexity is hidden instead of obvious. Depth of nesting is an input for the Rome Laboratory prediction model.

```
If a > 10 then
  If b < 5 then
    While not done
      ....
```