**Metrics Description**

In order to better measure selected projects, five different software measurement metrics are used. These five metrics belong to different aspects of software measurement. The details will be given as follow.

*Metric 1: Statement Coverage*

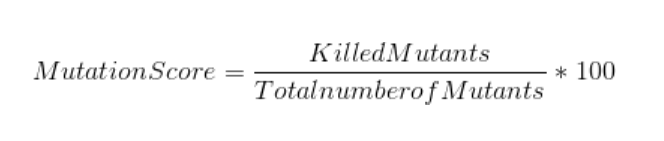
Test coverage is an important metric of software quality, since it indicates thoroughness of testing. In industry, test coverage is often measured as statement coverage [1]. According to the actual development experience, statement coverage is suitable for software measurement in projects analysis. Statement coverage count is how many statements are executed at least once during the test and thereby the more coverage percent it shows, the more opportunity to find the existing bug [2].

*Metric 2: Branch Coverage*

Though statement coverage is essential, it also has some defects. For example, statement coverage only consider the executed statements and ignore the combinations of branches. However, branch test is available to detect cryptic errors in code. As a result, branch coverage was chosen. Branch coverage is how many branches from each decision point is executed at least once thereby the more coverage percent it shows, the more opportunity to find the existing bug [2].

*Metric 3: Mutation Score*

To find weakness of code, mutation score is an useful measurement metric. Mutation score could be obtained through mutation testing. Mutation testing is a means of creating more effective test cases. Mutation testing is primarily used as a program-based technique. It uses mutation operations to mutate the program and generate program mutants. The goal in mutation testing is killing the generated mutants by causing the mutant to have different behavior from the original program on the same input data [3]. The way to calculate mutation score as follow:



*Metric 4: McCabe Metric (Cyclomatic Complexity)*

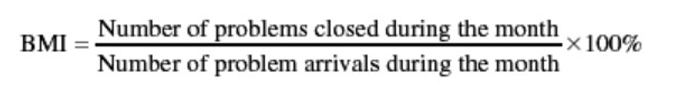
Complexity is vital to software measurement. In the analysis, McCabe complexity (Cyclomatic Complexity) was selected. Cyclomatic Complexity is used as indicators for program modularization, revising specifications, and test coverage. In addition, it has been used in software quality prediction models, whose purposes include predicting fault numbers through multivariate regression analysis and identification of error-prone modules based on discriminant analysis [4]. For calculating McCabe complexity, followed elements should be counted:

* E = the number of edges in CFG
* N = the number of nodes in CFG
* P = the number of connected components in CFG
* D = the number of control predicate (or decision) statements
* For a single method or function, P is equal to 1
* Cyclomatic Complexity = E – N + 2P

(Or Cyclomatic Complexity = D + 1)

*Metric 5: Fix Backlog and Backlog Management Index (BMI)*

In software measurement, software maintenance effort should not be ignore. Backlog and backlog management index is related to software maintenance effort and it is a metric to manage the backlog of open, unresolved. If BMI is less than 100, then the backlog increased. With enough data points, the techniques of control charting can be used to calculate the backlog management capability of the maintenance process. More investigation and analysis should be triggered when the value of BMI exceeds the control limits. A BMI trend chart or control chart should be examined together with trend charts of defect arrivals, defects fixed (closed), and the number of problems in the backlog [5]. The formula as follow shows:



*Metric 6: Maintainability Index*

Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of Lines of Code, Cyclomatic Complexity and Halstead volume. The calculation method as follows：

First, we need to measure the following metrics from the source code:

* V = Halstead Volume
* G = Cyclomatic Complexity
* LOC = count of source Lines of Code (SLOC)
* CM = percent of lines of Comment (optional)

From these measurements the MI can be calculated:

The original formula:

* MI = 171 - 5.2 \* ln(V) - 0.23 \* (G) - 16.2 \* ln (LOC)

The derivative used by SEI is calculated as follows:

* MI = 171 - 5.2 \* log2(V) - 0.23 \* G - 16.2 \* log2 (LOC) + 50 \* sin (sqrt (2.4 \* CM))

The derivative used by Microsoft Visual Studio (since v2008) is calculated as follows:

* MI = MAX (0, (171 - 5.2 \* ln (Halstead Volume) - 0.23 \* (Cyclomatic Complexity) - 16.2 \* ln (Lines of Code)) \*100 / 171)

In all derivatives of the formula, the most major factor in MI is Lines of Code, which effectiveness has been subjected to debate.