Software Metrics as Error Predictors
Mike Chapman and Daniel Solomon
NASA IV&V Facility, Fairmont, WV

Abstract
Analysis of a medium-size software development program shows that several common metrics (McCabe, Halstead, and operator/operand) can be said to predict errors. However, we show that this is largely because high metrics correlate with long modules, which tend to have more errors because they have more code. The use of metrics is not supported for the customary purposes of prioritizing reviews or requiring modules to be within a threshold.

Index Terms
Software metrics, complexity measures, quality, inspection

Introduction
It has been suggested [1-2] that modules exceeding a threshold value of cyclomatic complexity are difficult to test completely. It is reasonable to suspect that incompletely tested software may be delivered with errors. These considerations have suggested that developers be required to produce software whose cyclomatic complexity is below a given threshold, and/or that software reviews concentrate on software exceeding threshold. Similarly, folklore has it that IBM used to require that all software modules fit on two printed pages, the rationale being that longer subroutines would have more errors. A recent survey in [3] suggests several relations between errors and metrics.

To say that a metric is a good predictor of error conventionally means that software modules exceeding a threshold value of that metric average more errors than other modules. The goals of metric analysis are to help coders make fewer mistakes and to help inspectors find mistakes more efficiently. If you don’t have the resources to make careful inspections of all the custom code, it is hoped that the correct use of a software metric will help prioritize the software for review.

We show several ways of analyzing metrics and thresholds and find that per the conventional definition, all the metrics are good predictors of error for our data set. However, it turns out that the conventional definition is misleading. The use of metrics to identify and eliminate error-prone software may not reduce the total number of errors. This will be illustrated by considering detection and false alarm probabilities for the metrics. To understand why none of the metrics are particularly useful, we studied error density, rather than the number of errors. Error density does not correlate with any of the metrics. Modules with high values of metrics may have more errors simply because they have more code. Reliance on metrics is not warranted.

Definitions
A module is the smallest independent unit of source code. The following metrics are defined for each module.

- Lines of code, LOC, is a count of all the lines (source, blank, and comment) from the declaration line until the close bracket. This is not the same as SLOC, the number of executable lines of code.
- Cyclomatic Complexity, v(G), is the number of linearly independent paths through a module. This is an indication of how much effort is required to test a module, if the test plan is to supply diverse inputs so that all combinations of branches are executed. According to [2] modules with v(G) > 10 are at risk for reliability.
- Essential complexity, ev(G), is the cyclomatic complexity of the reduced flowgraph of a module after all structured constructs have been removed. Thus, ev(G) measures the degree to which a module contains unstructured constructs. Since structured code is easier to understand for subsequent modification, according to [2] modules with ev(G) > 4 are at risk for maintainability.
- Design complexity, iv(G), is the cyclomatic complexity of the reduced flowgraph of a module after all decisions and loops not containing calls to other modules are removed. Thus, iv(G) is a measure of the module’s decision structure as it relates to calls to other modules.
- Unique Operators, n1, is the number of distinct operators (such as +, -, etc.).
- Unique Operands, n2, is the number of distinct operands (such as variables).
- Total Operators, N1, and Total Operands N2, are the total number of operators and operands.
- Difficulty, D = n1 N2 / 2 n2 is the difficulty.
- Intelligence content, I = (N1 + N2)/D log2 (n1 + n2), is the complexity of the algorithm (theoretically independent of the coding language).
- Error Estimate, B = (N1 + N2) log2 (n1 + n2)/3000, is an a priori estimate of the number of errors in the module.

This study does not include the other five Halstead metrics, which have simple dependencies on the above metrics: Length, Volume, Level, Programming Effort, or Programming Time.

The data set
This software development project analyzed for this study consists of over 300,000 source lines of C code distributed
among well over 12,000 modules. We have eight years (the life of the project) of problem reports, including 1652 software problem reports whose fix included at least one change to a software module. A total of 4221 module changes were distributed among 2848 modules changed at least once due to a problem report. Any change in a module associated with a problem report we are calling an error for that module. Each problem report resulted in an average of 1.48 errors (module changes).

From this initial data set, we eliminated several subsets of data: a subsystem that had recently been redesigned and coded in C++, one subsystem consisting of multiple directories of essentially the same code, and 1007 errors whose modules no longer exist in the baseline or were ambiguous in their naming. The 12,476 surviving modules contained 503,460 (McCabe) LOC and 3157 changes due to error distributed among 2072 modules. The character of the data set is illustrated in Figures 1 and 2, which show the number of modules there are with each value of LOC and v(G). All of the metrics exhibit similar distributions. Note there are very few modules with very high values of any of the metrics (often just one).

Error rate and probability of error
Given a set of modules, the error rate, ER, is the average number of errors among modules in the set. We can examine the relationship between ER and each of the metrics by calculating the ER of the modules having a given value of each metric. The resulting smoothed ER curves are shown for LOC and v(G) in Figures 3 and 4. The other metrics have similar curves. Error rate increases with increased values of all the metrics. Several conclusions can be drawn:
1. If a module has a very high metric, it probably has many errors.
2. If a module has a very low metric, it probably does not have many errors.
3. For the first two conclusions, it doesn’t matter much which metric is used.
So far, it appears that the metrics are appropriate as predictors of error.

To use a metric for prioritizing reviews or requiring modules to be within a threshold, a threshold must be chosen. Then the problem surfaces. According to the McCabe software documentation [2], modules with v(G) >10 or ev(G) > 4 are potentially problematic. As can be seen in Figure 4, v(G) = 10 is bad enough: among all modules with cyclomatic complexity equal to 10, there is an average of one error for every three modules. In fact, there are 1912 errors (of the 3157) in modules with v(G) ≤ 10. Thus, the selection of 10 as a threshold means one would miss 60% of the errors.

1 The data used for this project was made available to the IV&V Facility Metrics Data Program and can be obtained from mdp.ivv.nasa.gov.

2 For increasing values of the metrics, increasingly many values were averaged together; for example, the data point (84, 0.5) in Figure 3 indicates that among all 69 modules with LOC in the range 82 to 86, the average number of errors is 0.5.
Similar analysis holds for other metrics; e.g., for essential complexity, 60% of the errors are in modules with ev(G) ≤ 4.

For a set of modules, the probability of error, PE, is the number of modules in the set with at least one error divided by the total number of modules in the set. We can examine the relationship between PE and each of the metrics by calculating the PE of the modules having a given value of each metric. Figures 5 and 6 give the PE curve for LOC and v(G), with the same smoothing as before. The other metrics have similar curves. The PE data supports the same conclusions as the ER data above. Note, however, that 24% of modules with cyclomatic complexity = 10 have at least one error. In fact, there are 1458 modules with errors and v(G) ≤ 10. Thus, the selection of 10 as a threshold means one would miss 70% of the modules with errors. Similar analysis holds for other metrics; e.g., for essential complexity, 68% of the modules with errors have ev(G) ≤ 4.

**Probability of detection and false alarm**

These concepts from radar theory provide insight into metric use to identify modules likely to contain errors. For a given threshold of a given metric, define

- PD = number of errors in modules above threshold divided by the total number of errors.
- PFA = number of modules above threshold without errors divided by the number of modules without errors.

Figures 7-12 show plots of PD vs. PFA for the various metrics. Each plotted point represents (PFA, PD) using various threshold values. As the threshold decreases, PD and PFA both increase. The perfect detector would have a point at the upper left hand corner, indicating 100% detection and no false alarms. No threshold values of any of our metrics give PD above 60% without PFA at least 20%. Similarly, to have PD above 80%, the PFA is above 40%. Reliance on any metric to determine which software to review will result in missing a significant fraction of the errors, as well as reading many modules without errors.
Error density

Normalizing the data by considering error density (ED) removes the effect that longer modules have more errors simply because they have more code. For a set of modules, ED is defined as 1000 times the number of errors in the set divided by the total LOC of the set. Figures 13-15 show ED for LOC, v(G) and unique operands (\(n_2\)). We noted in Figure 3 that longer modules have more errors, but we see in Figure 13 that the ED actually decreases with increased module length. This suggests that requiring modules to be short (and more numerous) would result in more total errors distributed into more modules. The complexity, intelligence, and error estimate data show no trend. ED decreases as the operator, operand, or difficulty increase.

Figure 13 Average Error Density vs. LOC
Figure 14 Average Error Density vs. v(G)
Figure 15 Average Error Density vs. Unique Operands

Figure 16-17 show the ratio of the ED of modules above threshold to the ED of modules at or below threshold for LOC and complexity. The other metrics have similar curves. The plots indicate that modules above threshold do
not have larger ED than modules below threshold. Again, prioritizing modules for review based on one of these metrics is not indicated, since there are as many errors per LOC in modules below (any chosen) threshold.

Conclusions
Contrary to the conventional wisdom, neither longer nor more complex software modules appear to have higher error density. While higher values of a metric are associated with more errors, so that these metrics may be said to be predictors of error, error density suggests that this correlation may not be helpful. In particular:

- Requiring coders to write low-metric modules would apparently only succeed in having more errors distributed into many shorter (low-metric) modules.
- Relying on metrics to determine which software to review will result in missing a significant fraction of the errors, as well as reading many modules without errors.

Future work
The authors plan to carry out similar analysis for additional software development projects to see whether the present conclusions are more widely supported.

Acknowledgements
The authors wish to acknowledge the support of the IV&V Center Software Initiative Research Program.

References