OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

A Note on the Style Metric of Berry and Meekings

Warren Harrison
School of Business Administration
University of Portland
Portland, Oregon 97203

Curtis Cook
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

85-60-3

A NOTE ON THE STYLE METRIC OF BERRY AND MEEKINGS

Warren Harrison
University of Portland
Portland, OR 97203
USA

Curtis Cook
Oregon State University
Corvallis, OR 97331
USA

ABSTRACT

The "style metric" of Berry and Meekings is purported to quantify the lucidity of software written in the C programming language. We used a modification of this metric to try and identify error-prone software. Our results indicate that this metric seems to bear little relationship to the density of errors found in programs.

CR Categories and Subject Descriptors:

D.2.8 [Software Engineering]: Metrics - complexity measures;

D.2.9 [Software Engineering]: Management - software quality assurance

General Terms:

Languages, Measurement

Additional Key Words, Phrases:

C, style analysis

INTRODUCTION

Programming style is an elusive and intuitive quality of a program. Schneider and Buell [1, p 52] define programming style as "the entire set of conventions, guidelines, aids and rules that make computer programs easier for people to read, work with and understand". Numerous books and articles such as [2,3] suggest and list style rules for various programming languages.

Programming style is difficult to quantify. Several software complexity metrics (Halstead's effort E [4], McCabe's cyclomatic complexity [5], and the number of lines in a program are the most popular) have been used as measures of program clarity in programming style studies. The results have been mixed. Gordon [6] found a corresponding decrease in E, while Evangelist [7] found all the metrics insensitive to the application of style rules and inconsistent in rewarding programs written according to generally accepted style rules.

Barry and Meekings [8] defined a style metric based on how closely a program conforms to a set of style rules. They defined a style score ranging from 0 to 100 as the weighted sum of 11 program characteristics. A score of zero is poor, and 100 is excellent.

The characteristics (and their weights) comprising this metric are: module length (15), identifier length (14), percentage of comment lines (12), percentage of indentation

spaces to all characters (12), percentage of blank lines (11), average number of nonblank characters per line (9), average number of spaces per line (8), percentage of symbolic constants used (8), number of reserved words used (6), the number of #INCLUDE files used (5) and number of GOTOs used (-20). They presented a table from which the percentage of the maximum score could be computed for each characteristic.

The eleven characteristics, the contribution of each characteristic, and the table for computing the score for each characteristic was based on Rees [9]. Rees based his selection of characteristics on intuition, experience, and ease of implementation, and considered only the influence of layout and identifiers on style.

Berry and Meekings feel their style score measures the lucidity, or understandability of a program. When they applied their style metric to a large collection of programs written by experienced programmers (C programs for the UNIX [trademark of Bell Labs] operating system), they discovered a great disparity of style scores. They attributed this disparity to sparse commenting and blank lines, and the tendency of C to encourage "concise programs".

Berry and Meekings analyzed a collection of C programs and did not attempt to determine the relation between their characteristics or weightings, and program clarity. In this

paper, we consider the latter problem. We felt that if this metric does indeed measure program understandability, then it could be used to identify programs which may be potential trouble spots in a system.

A piece of code could be a "trouble spot" due to error proneness. We consider error proneness to be the number of errors in the program, normalized in some manner by the program's length. We refer to this as "error density", or the average number of executable lines of code between error occurances. It seems that the more error prone a program, the more likely some errors will not be found until after it is delivered. It can be argued that lack of understandability contributes to this situation. Thus, by measuring the understandability of a piece of software programs which are highly error prone may be identified. We hoped to determine if programs that possess high style scores were any less error prone than other programs which did not.

THE DATA

To determine if the style metric could distinguish among programs of differing error proneness, we analyzed 20 multi-function C modules. These modules were taken from a language/environment project carried out by a large Fortune 500 corporation. The data analyzed represented over 35,000 lines of C code. The data was made available to us in its Reduced Form

[10], so every aspect required for the Berry and Meekings metric was not available.

The Reduced Form is a technique which facilitates the distribution of software characteristic data for complexity metric studies by industrial organizations. It provides information on most of the "essential characteristics" (at least in terms of complexity metrics) of a piece of software, while barring unauthorized reproduction of the code. Since developing this technique, we have had access to many software systems which before were not available to us.

However, in return for almost total security, the Reduced Form does not provide information about some characteristics of the software which might be useful (but which, at the same time, may disclose certain information about the software which could be proprietary). For example:

- Variable names are aliased, so length of variable names are not available.
- All "textual characteristics" of the actual code are hidden, so line length, indentation and embedded spaces are not accessible.
- The idea of number of blank lines and comment lines are combined to arrive at a figure representing the number of non-executable lines.

However, we felt that enough of the characteristics used in

Berry and Meekings' metric were available to investigate the relationship between the programming style metric and error proneness. The modified metric we used is shown in Table I.

As can be seen, this includes only four components. One of these components (X2) combines two of the original metric's components (percentage of comment lines and percentage of blank lines). Identifier Length, Indentation, Characters Per Line, and Spaces Per Line were not available. However, as Berry and Meekings point out, three of these characteristics could easily be affected by the use of a program formatter. No GOTO statements were found in any of the code we analyzed, so we deleted this factor from the caclulation for simplicity.

We felt that if the original metric were an accurate measure of program "style", the use of a subset of the metric would also be a useful gauge of style (albeit, a less refined one).

In addition to the Reduced Form data describing each module, the number of errors encountered during system testing was also made available to us. The characteristics and number of errors for each module are shown in Table II.

RESULTS OF THE ANALYSIS

In order to determine the relationship (if any) that held between the style metric and the error proneness of each module, we performed a simple correlation analysis [11]. The results of our correlation analysis were discouraging. It appears that a

correlation of only -.052 existed between the observed error density and the style metric. This suggests that the style metric bears no relationship to the error density encountered in a group of programs (at least based on our data).

CONCLUSIONS

Two problems of any programming style metric based on conformity to a set of style rules are:

(1) Adequacy and completeness of the style rules;

(2) Measuring the degree of conformity.

Berry and Meekings style metric was based on a weighted sum of eleven prorgam characteristics. It certainly is not clear that their eleven characteristics comprise an adequate, let alone complete, set of style rules. Their weightings reflect their intuition about the degree of conformity. Our results show that although Berry and Meekings' metric is a promising beginning, the relation between style rules and program characteristics warrants much additional study.

Since the goal of programming style is to aid comprehension, a programming style metric should measure the effect of style rules on comprehension. In particular, the contribution to understanding of the individual style rules, such as header and paragraph comments, mnemonic variable names, indenting and formatting, should be investigated. Rather than just determining whether an individual style rule affects comprehension, these

investigations should seek to answer the question how it does or does not facilitate understanding.

REFERENCES

[1] Schneider, G. and S. Buell, Advanced Programming and Problem Solving With Pascal, John Wiley, New York, 1981.

[2] Kernighan, B. and P. Plauger, The Elements of Programming Style (second edition), McGraw-Hill Book Company, New York 1978.

[3] Marca, D., "Some Pascal Style Guidelines", ACM SIGPLAN Notices, April 1981, pp 70-80.

[4] Halstead, M., Elements of Software Science, Elsevier North-Holland, New York, 1977.

[5] McCabe, T., "A Complexity Measure" IEEE Transactions on Software Engineering

[6] Gordon, R., "Measuring Improvements in Program Clarity", IEEE Transactions on Software Engineering, Vol SE-5, 2, March 1979, pp 79-90.

[7] Evangelist, M., "Program Complexity and Programming Style", Proceedings International Conference on Data Engineering (1984), pp 534-541.

[8] Berry, R. and B. Meekings, "A Style Analysis of C Programs", Communications of the ACM, 28, 1 (January 1985), pp 80-88.

[9] Rees, M., "Automatic Assessment of Aids for Pascal Programs", ACM SIGPLAN Notices 17, 10 (October 1982), pp 33-42.

[10] Harrison, W. and C. Cook, "A Method of Sharing Software Complexity Data", to appear.

[11] SPSS, SPSSx User's Guide, McGraw-Hill Book Company, New York, 1983.

Let X1 represent the average length, in executable lines, of function definitions in a module

```
if 10 < X1 < 25, then P1 = 36.6
if  4 < X1 < 10, then P1 = X1 * 6.1 - 24.4
if 25 < X1 < 35, then P1 = X1 * -3.7 + 129.5
```

Let X2 represent the average percentage of nonexecutable lines of the function definitions in a module

```
if 15 < X2 < 25, then P2 = 29.3
if  8 < X2 < 15, then P2 = X2 * 4.2 - 33.6
if 25 < X2 < 35, then P2 = X2 * -2.93 + 102.5
```

Let X3 represent the average percentage of symbolic constants in the function definitions of a module

```
if 15 < X3 < 25, then P3 = 29.3
if 10 < X3 < 15, then P3 = X3 * 3.9 - 39
if 25 < X3 < 30, then P3 = X3 * -3.9 + 117
```

Let X4 represent the average number of reserved words per function definition of a module

```
if 16 < X4 < 30, then P4 = 14.6
if  4 < X4 < 16, then P4 = X4 * 1.2 - 4.8
if 30 < X4 < 36, then P4 = X4 * -2.4 + 86.4
```

```
SCORE = P1 + P2 + P3 + P4
```

Table I. The alternative style metric. This metric was adapted from the original metric of Berry and Meekings. Since the data was provided in its Reduced Form, every component of the original metric was not available to us. We determined the characteristics which were available, and adjusted their contribution to the total SCORE proportionally, so that they would total to 100.

	NCSL	PRC	X1	DSL	X2	CON	Op'rnd	X3	X4	SCORE	BUGSD
1	1137	31	37	1412	.19	68	718	.09	24	43.90	162
2	2406	51	47	3299	.27	10	1053	.01	23	37.79	71
3	333	20	17	485	.31	6	147	.04	10	53.93	333
4	879	26	34	1098	.20	27	397	.07	18	43.90	88
5	785	36	22	1083	.28	19	468	.04	16	72.81	131
6	1522	3	507	2805	.46	0	63	.00	20	14.60	127
7	1591	4	398	2569	.38	0	168	.00	25	14.60	76
8	639	1	639	1214	.47	0	20	.00	17	14.60	53
9	134	2	67	207	.35	0	51	.00	31	13.20	134
10	1568	25	63	2318	.32	19	505	.04	24	22.30	523
11	1251	22	57	1772	.29	13	447	.03	26	30.95	313
12	554	24	23	681	.19	3	311	.01	18	80.50	185
13	779	43	18	1263	.38	0	225	.00	10	43.77	52
14	2927	87	34	4497	.35	97	1069	.09	18	14.81	113
15	2605	76	34	3058	.15	69	1365	.05	20	43.22	69
16	816	29	28	1098	.26	21	369	.06	19	41.85	91
17	851	20	43	1209	.30	42	328	.13	17	41.28	284
18	747	31	24	992	.25	9	562	.02	16	80.50	57
19	2954	84	35	3725	.21	40	1548	.03	22	43.90	134
20	2716	103	26	3505	.23	41	1643	.02	23	43.90	160

Table II. Characteristics of modules, style score and error density.