Aesthetics Versus Readability of Source Code

Ron Coleman
Computer Science Department
Marist College
Poughkeepsie, NY, 12601, United States

***Abstract*—The relationship between programming style and program readability has never been examined empirically, although the association has substantial importance for both pedagogical and industry best practices. This paper studies a fractal, relativistic measure of programming style called the beauty factor or "beauty" and puts forward two new hypotheses of beauty. First, code with increasing beauty tends to be more readable. Second, beauty measures a unique property in code called aesthetic value distinct from readability. These hypotheses are tested on a corpus of 53,000 lines of open source system codes written by experienced Linux programmers. Statistical correlation analysis is used on 11 different beauty factors versus eight different readability models (i.e., 88 experiments total). As the primary finding, the data show the maximum absolute statistically significant correlation is $|\rho|=0.59$ whereas the absolute median correlation is $|\rho|=0.33$. In other words, at least 65% of statistically significant variations in beauty cannot be explained by variations in readability; approximately 90% of statistically significant variations in beauty cannot be explained typically by variations in readability. These results lend support to both hypotheses. The data further shows indentation is more reliably correlated with readability than mnemonics or comments and GNU style is more correlated with readability than K&R, BSD, or Linux styles.**

Keywords—Programming style; fractal geometry; readability

"We have seen that computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty." ~Donald E. Knuth, 1974 ACM Turing Award Lecture [1]

I. Introduction

In 1979 AT&T Bell Labs released Unix version 7 which included *cb*, the C beautifier [2]. A unique tool and the first of its kind, *cb* "beautified" programs, that is, reformatted them, according to rules prescribed by the K&R style [3]. In creating *cb*, software engineers did not address the ontology of *what is* beauty in code. They instead focused on the epistemology of *what is knowable* about such beauty which could be automated, demonstrated, practiced, and taught. They had hoped paying attention to sensori-emotional or aesthetic values in code might also promote readability of code. However, they were not explicit about this and left the precise nature of the relationship between aesthetics and readability assumed and open. The literature frequently confounds aesthetics and readability as if they are interchangeable (see for instance [4-6]). Indeed, the long-held, widely taught, and often repeated justification for good style is to make programs more readable and presumably, more maintainable [7].

One problem is that this view has never been tested empirically. Another problem is ontological. Readability is about understanding code and aesthetics is about appreciating it, *l'art pour l'art*. Thus, in principle, conflating aesthetic and readability is a category mistake. Finally, how different styles affect cogntive ease or difficulty of grasping code remains poorly understood. In other words, programmers may assume on the basis of preference, experience, etc., that one style is more readable than another but is personal taste as such supported by the data? The question has never been investigated empirically.

This paper studies a relativistic, fractal measure of programming style called the beauty factor (or "beauty" for short) and puts forward two new hypotheses. First, code with increasing beauty tends to be more readable. Secondly, beauty measures a unique property of code, call it aesthetic appeal, which is distinct from readability. To test these hypotheses, the paper assesses the beauty and readability scores of a statistically meaningful number of files and analyzes the correlations. No correlation denies the first hypothesis. Strong correlation denies the second hypothesis. Thus, both hypotheses can be logically true simultaenously only with weak or moderate correlation. In other words, the hypothesized aesthetic value of code is not necessarily completely orthogonal to nor a proxy for readability but a spectrum of potential.

II. Related Work

Kokol, et al, [8-10] showed that programs contain long-range correlations in characters and tokens. These researchers were searching for a fractal metric of software complexity using lexical analysis of a small sample of randomly generated Pascal programs. This paper uses a larger corpus of production C codes and image analysis. Coleman and Gandhi [11] hypothesized programming style might be related to fractals since fractals are often associated with beauty [12]. This effort showed that changes in style were systematically correlated with changes in fractal dimension. Coleman and Gandhi [13] proposed a fractal, relativistic model and showed beauty was weakly to moderately correlated with software complexity in directions that comported with style recommendations. In other words, there is virtue in brevity and simplicity in code. Coleman and Boldt [14] investigated disorder that is often introduced in code through maintenance and showed beauty was weakly to moderately anti-correlated with entropy. Coleman and Rahtelli [15] showed the beauty model predicts aesthetic value in scientific libraries; in this paper we study system codes. These investigations of beauty resemble efforts by researchers who used fractal geometry to assess aesthetic

values in paintings, including Pollock's "action paintings" [16-20]; the main technical differences are that this paper studies code as opposed to fine art and programming style as opposed artistic style. *Beautiful Code* [21] deals with conceptual beauty in the design and analysis of algorithms, testing, and debugging, topics which are outside the scope of this paper. Also outside the scope of this paper is the more general topic of program comprehension, which focuses on cognitive models to explain how programmers understand code and the development of tools to aid them [22, 23].

III. THEORY

A. Aesthetic Theory

1) Working Definitions

The subject of aesthetics has a long history of thoughtful consideration of matters of style and taste dating to ancient. According to the Stanford Encyclopedia of Philosophy the term "aesthetic" refers to, among other things, a kind of heuristic judgement or value called beauty, appeal, appreciation, virtue, goodness, etc. [24] We use the term "aesthetic appeal" when referring sensori-emotional judgements as opposed to other forms of appeal. Thus, the working definition of "beauty," namely, the measure of programming style, is a special case of the aesthetic definition. We say "beautiful" code has measurably more style than indecorous code. To "beautify" code is to measurably improve its style according to what we know about well-written code in terms of layout and structure. To "debeautify" code is to do the opposite; in fact, there are tools, known as "obfuscators" that use anti-style techniques usually for information security purposes [25, 26].

2) Immediacy and Disinterest theses Applied to Code

The immediacy thesis maintains that judgements of appeal are immediate or straightforward through sensory discernment. [24] For code, this suggests the possibility of appreciating its form as a work of art, that is, without attempting to "read" or "understand" its function. This paper argues (see below) that the rules of judging good programming style in this way are widely known and firmly established. So much that these rules can be automated by programs like *cb* which reformat target code without regard for how the target works or even what it does. What matters foremost, aside from preserving the semantics, is how the target looks in the end.

The disinterest thesis claims that judgements of appeal are not self-interested. This is the sense of "art for its own sake." For code, a programmer could appreciate the rules of a style like K&R for "intellectual and emotional satisfaction," as Knuth described it. [1] Whether K&R or one of the other styles we study makes the code more readable and how much is the subject of this paper.

3) Basic Tenets

The beauty model presupposes we know or can know epistemologically what programmers think—or better, how they *feel*—about style. This knowledgebase already exists in a mature and rich form that can be observed directly, repeatedly, and systematically in style guides, coding standards, organizational coding policies, textbooks, research reports, example codes, blogs, etc. It can also be observed indirectly through use and side-effects of tools like *cb*, functions embedded in modern IDEs for reformatting code code, and online sites that have codified this knowledge for a variety of styles and languages. In an observational study, Coleman and Gandhi [13] surveyed this knowledgebase and identified three general principles they called "basic tenets" of good programming style: namely, 1) use white space judiciously; 2) choose mnemonic names; and 3) include documentation.

4) Beauty Factors

Let S be some source called the control or "baseline." Then, we have S' such that

$$S' = T(S) \tag{1}$$

where T is a semantic-preserving transformation or treatment. That is, S and S' differ only in style. There are two modalities of T with respect to the basic tenets: beautification and debeautification. We encode S and S' separately as an in-memory bitmap called an *artefact*. (The fact that it is in-memory only serves to say it is independent of file format althought it may for some reasons reside in secondard storage.) Finally, we measure the fractal dimension of these artefacts using reticular cell counting (a.k.a., the box counting dimension), D [27]:

$$D(S) = \lim_{r \to 0} \frac{\log N_r(S)}{\log 1/r} \tag{2}$$

where $N_r(S)$ is the number of components (i.e., cells) covered by the ruler of size r. (Note that S in the above equation refers to the artefact of the baseline and not the baseline itself; we use this form only to simplify the notation.) Thus, $D(S)$ is the slope of the regression line over different ruler sizes, r. We similarly measure $D'=D(S')$. The beauty factor model, B, is given by the following equation:

$$B(S \mid T) = k \log (D / D') \tag{3}$$

where k is a constant. When k=10 and the logarithm is base ten, the units are decibels. B is indicated as follows:

1. If B < 0, the style of S might be improved by T.
2. If B ≥ 0, the style of S probably won't be improved by T.

A *contrary indication* implies T is categorized in a modality mathematically with $-B$.

5) Semantic-Preserving Transformations

The tables 1 and 2 give the semantic-preserving treatments we use in this paper.

TABLE I. BEAUTIFYING TREATMENTS

T	Tenet	Regime
GNU	1	Apply GNU style [28].
K&R	1	Apply K&R style [3].
BSD	1	Apply BSD style [29].
LIN	1	Apply Linux style [30].
MNE	2	Refactor names to be more mnemonic
REC	3	Add one or more comments.

TABLE II. DE-BEAUTIFYING TREATMENTS

T	Tenet	Regime
NOI	1	Remove indents.
R2	1	Randomize indent with 1-2 spaces.
R5	1	Randomize indent with 1-5 spaces.
NON	2	Refactor names to be less mnemonic
DEC	3	Remove all comments.

These transformations are not, nor intended to be, exhaustive. Rather, they are indicative of transformations in general and sufficient to test the main hypotheses.

6) Block Artefact Method

The literal artefact method (or LAM) encodes the bitmap using a fixed width font as S (or S') literally looks like itself. The figure 1 shows the file, hello.c, with its LAM encoding. We do not use LAM in this paper.

```
#include <stdio.h>
int main(int argc, char** argv)
  printf("Hello, world!");
  return 0;
}
```

Fig. 1. LAM encoding of hello.c

The block artefact method (or BAM) which we use in this paper encodes the bitmap with block characters (e.g., ■) in place of the regular, textual characters and tabs. It leaves spaces as blanks. The figure 2 shows hello.c with its BAM encoding

Fig. 2. BAM encoding of hello.c

From a fractal point of view, LAM and BAM are strongly correlated with r=0.95 and thus, only one of them is needed for the purposes of this paper [10]. The primary advantage of BAM over LAM is the former is more robust against language dependencies, both programming and the cultural languages in the case of names and comments. Furthermore, BAM effectively destroys the source readability in favor of the text's spatial-visual pattern in direct support of the immediacy thesis. We use only BAM in this paper.

B. Readability Theory

The literature on program understanding presupposes the existence of a posited cognitive load associated with comprehending code through an understanding hierarchy. The "bottom-up" theory of understandability maintains that the first step in grasping the code is to read it. Higher mental models of program flows, organization themes, abstraction, design patterns, etc. develop from this first and arguably, necessary step. Readability models hypothesize that the cognitive load is measurable and furthermore, correlated with density of operators and operands, logic complexity, lines of code (LOC), statement length, number of statements, etc. Advocates apprehend these quantities as constituting metrics they define as "readability". While it is clear that readability models reside at the lowest level in the understand hiearchy, they do not, nor are they designed to, capture all aspects of program understanding. According to Borstler, et al [31], readability models have value precisely because they "catch" some aspects of program understanding rather than attempting to measure it entirely.

Readability research is perhaps represented variously in the literature by three distinct generations: Halstead, machine-learned, and prose-inspired. They are similar in that they each use syntactic featuires (e.g., line length, number of identifiers, length of identifiers, etc.) to assess the readability of source.

1) Halstead Statistics

Halstead [32] was interested in predicting programming effort to which he related physical quantities like volume and gas pressure. He defined *program length*, *N*, to be the sum of operators, N_1, and operands, N_2,

$$N = N_1 + N_2 \tag{4}$$

Program vocabulary is the sum of unique is the sum of unique operators, n_1, and unique operands, n_2,

$$n = n_1 + n_2 \tag{5}$$

Program volume is a measure similar to Shannon entropy, namely,

$$V = N \log n \tag{6}$$

Difficulty combines information about unique operators and operands and total operations,

$$Df = N_1 \, n_1 / \; 2 \, n_2 \tag{7}$$

Effort is difficulty as a multiplier of volume:

$$E = Df \times V \tag{8}$$

A higher value in any of these statistics predicts the code is more difficult to read.

2) Machine-Learned Metrics

Buse and Weimer (or BW) [33] used supervised learning to train a Bayesian classifer to associate Halstead-type measurements with human judgements of the same code. After training is complete, the classifier operation is $X \rightarrow Y$ where X is a code snippet and Y is its predicted readability score.

Posnet, Hindle, and Devanbu (or PHD) [34] endeavored to simplify and improve the BW model using classical software engineering and information theory defined as

$$PHD = 1 / (1 + e^{-z}) \tag{9}$$

where

$$z = 8.87 - 0.033 \, V + 0.40 \, LOC - 1.5 \, H \tag{10}$$

where V is the Halstead volume, LOC is lines of code, and H is the Shannon information of tokens, namely,

$$H(S) = -\sum_{i=1}^{n} p_i \log p_i \tag{11}$$

and p_i is the fraction of tokens, s_i, in a source, S. In this case, the learning was derived through regression analysis.

A higher value of BW or PHD predicts the source code is more difficult to read.

3) Prose-Inspired Metrics

These metrics are inspired by Flesch-Kincade [35] readability score for prose. Abbas [36] with Borestler, Cspersend, and Nordstrin [31] developed the "average sentence length" (ASL) metric which is the average number of tokens per statement. They also formulated the "average word length" (AWL) metric, which is the average length of tokens adusted for special operators like ".". The authors integrated these measures into a model they called the "software readability ease score or SRES,

$$SRES = ASL - 0.1\ AWL \quad (12)$$

A higher SRES value predicts the source code is more difficult to read.

4) Readability Indications

The table 3 summarizes the readability indications each model.

TABLE III. READABILITY MODEL INDICATIONS

Model	Predicts
N	Difficulty
n	Difficulty
V	Difficulty
Df	Difficulty
E	Difficulty
BW	Ease
PHD	Ease
SRES	Difficulty

IV. METHODS

The experimental design is conceptually very simple: generate the beauty and readability scores for each file in the corpus and using robust methods, analyze correlations.

A. C language

The choice of C was motivated by a few factors, one being that C is one of the most widely used languages, frequently at or near the top of popularity and for a number of years [37, 38]. Another is many modern language designs have been significantly influenced by and borrow from C and consequently, what may be valid for C could have implications broadly for other languages. Finally, C has a production-quality beautifier tool, *indent* [39], which succeeds *cb*. It generates GNU, K&R, BSD, and Linux styles in Table 1.

B. Corpus

The GNU Core Utilities [40] is the testbed. It consists of 114 open source programs that comprise of the standard shell commands like *cat*, *ls*, *sort*, etc. of the Linux operating system. In total, there are nearly 70,000 LOC. However, we don't use the C programs directly. We instead follow Coleman and Gandhi [14] and strip compiler and preprocessor directives, prototypes, typedefs, copyright notices, and the like, then decompose the remains into 1,043 single-function C files or approximately 53,000 LOC. Essentially, anything outside a function definition gets filtered except for comments that immediately precede the function definition; we keep those. We believe single-function C files as such reduces the number of confounding variables and simplifies the study. Furthermore, single-function C files lend the experimental results greater generalization for languages like Java, Python and others that are similar to C except in those aspects we remove.

The figure 3 (Chart) below gives summary statistics of the corpus of 1,043 baselines. The minimum length is 2 LOC; the median is 32 LOC; and the maximum is 1,034 LOC.

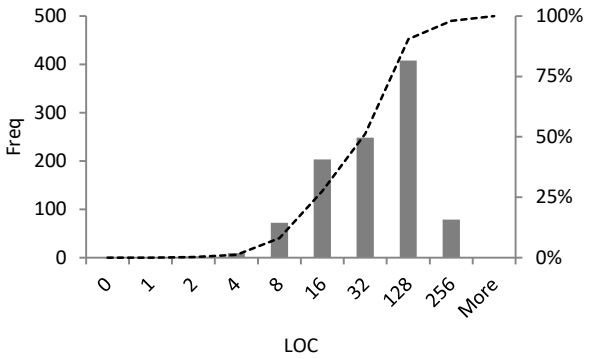

Fig. 3. Distribution of baseline sizes of the test bed corpus.

C. Fractal Dimension

To estimate the fractal dimension we repurpose a library, Fractop [41], which had been originally designed to estimate the fractal characteristics of nerve tissue images. For ruler, *r*, in Equation 2, we use 2, 3, 4, 6, 8, 12, 16, 32, 62, and 128 pixels which is the default grid in Fractop.

D. Result Matrices

There are 11 beauty factors and eight readadbility scores, respectively. For each file in the corpus, calculate these 19 scores. Since we are studying only aesthetics versus readability, not aesthetics by itself nor readability by itself, we only compute the correlation coefficients to generate the 11×8 matrix. However, for simplicity, we present the results in three separate, grayscale-encoded matrices: 11×5 for the Halstead models, 11×2 for the machine-learning models, and 11×1 for prose-inspired model. We intend the use of grayscale only to explicate and explain general patterns in the data.

E. Statistical Methods

Preliminary analysis of results using the Kolmogorov-Smirnov test of normality suggest the distributions of beauty factors and readability scores are not Gaussian. Thus, we use instead Spearman's rho (ρ), the rank-based correlation coefficient. Since the degree of correlation is important for our study, we use widely accepted definitions of "weak", "moderate" and "strong" correlation per the table 4.

TABLE IV. CORRELATION DEFINITIONS WITH RANGES, UPPER-BOUND P-VALUES, AND GRAYSCALE INDICATOR SHADES

Correlation	$\lvert\rho\rvert$	$\lceil P$-value $\rceil$	Indicator
Strong	[0.70, 1.00]	$< 10^{-153}$	
Moderate	[0.30, 0.70]	$< 10^{-22}$	
Weak	[τ, 0.30]	<0.05	(clear)
None	[0, τ]	≥0.05	

where τ=0.051 for the one-tailed test (i.e., where direction matters) versus τ=0.061 for the two-tailed test (i.e., where direction doesn't matter).

According to our hypotheses, we expect readability ease (see Table 3) to be positively correlated with beautification and negatively correlated with debeautification. We expect readability difficulty to be negatively correlated with beautification and positively correlated with debeautification. If beauty is contraindicated, then these correlations are reversed.

F. Code Analysis Code

The codes that analyze the corpus for beauty and readability are freely available on GitHub.com [42]. The remainder of statistical methods are implemented in Microsoft Excel.

V. Results

This section gives the results of experiments.

A. Beauty versus Halstead metrics

The table 5 gives rank correlations of beauty with Halstead statistics.

TABLE V. Spearman's Rank Correlation Coefficients of Beauty Vs. Halstead Statistics

B		Halstead statistics				
Modality	T	N	n	V	Df	E
Debeautify	NOI	-0.55	-0.55	-0.56	-0.53	-0.56
	R2	0.15	0.15	0.15	0.14	0.15
	R5	0.10	0.10	0.10	0.09	0.10
	NON	0.48	0.47	0.48	0.49	0.50
	DEC	0.18	0.18	0.18	0.24	0.21
Beautify	GNU	-0.58	-0.57	-0.59	-0.56	-0.59
	K&R	-0.46	-0.45	-0.46	-0.46	-0.47
	BSD	-0.46	-0.45	-0.46	-0.37	-0.43
	LIN	-0.51	-0.50	-0.51	-0.53	-0.53
	MNE	-0.31	-0.28	-0.30	-0.31	-0.31
	REC	0.09	0.07	0.09	0.10	0.09

B. Beauty Versus Machine-Learned Metrics

The table 6 gives rank correlations of beauty with machine-learned statistics.

TABLE VI. Spearman's rank correlation coefficients of beauty vs. machine-learned metrics

B		Machine-learned metrics	
Modality	T	BW	PHD
Debeautify	NOI	0.36	0.33
	R2	-0.12	-0.12
	R5	-0.05	0.01
	NON	-0.37	-0.36
	DEC	-0.18	-0.07
Beautify	GNU	0.44	0.31
	K&R	0.31	0.31
	BSD	0.38	0.29
	LIN	0.29	0.38
	MNE	0.17	0.18
	REC	-0.06	-0.00

C. Beauty Versus Prose-Inspired Metric

The table 7 gives rank correlations of beauty with prose-inspired statistics.

TABLE VII. Spearman's rank correlation coefficients of beauty vs. prose-inspired metric

B		Prose-inspired metric
Modality	T	SRES
Debeautify	NOI	0.38
	R2	0.04
	R5	0.02
	NON	0.29
	DEC	0.05
Beautify	GNU	-0.44
	K&R	-0.35
	BSD	-0.41
	LIN	-0.29
	MNE	-0.16
	REC	0.04

VI. Discussion

We summarize patterns in the data in a series of points. Note that in every case except where explicitly noted, the results are statistically significant with the ceiling of corresponding *P*-values in Table 4.

A. Prevalence of Weak-to-Moderate Correlations

The data in Tables 5-7 clearly beauty and readability are related in statistically significant ways. The correlations are all weak or moderate (i.e., the cells are clear or shaded as ■; there are no cells shaded as ■.) There is one There are a few cases of no correlation but these are not statistically significant. These data support our first hypothesis.

Furtheremore, beauty and readability are correlated in the directions we would expect with the exception of NOI and REC (see below). For instance, in Table 5 the de-beautifying treatments are positively correlated with Halstead statistics while all the beautifying treatments are anti-correlated with Halstead statistics. A similar pattern exists in Table 6—and appropriately since SRES, like the Halstead statistics, indicate difficulty of readability. In Table 7, we note the opposite pattern: BW and PHD indicate ease of readability and they are thus, positively correlated with beauty factors.

B. Absence of Strong Correlations

Tables 5-7 show no evidence of strong correlations. If we ignore direction, the range of statistically significant correlations is $|\rho|=[0.05,0.59]$. Thus, according to R^2 analysis, at least 65% of variations in beauty cannot be explained by variations in readability. However, the median correlation which is more representative, $R^2=0.0961$. In other words, less than 10% of variations in beauty can be explained by variations in readability. These data support our second hypothesis: beauty is a unique property in so far as readability is concerned. In other words, beauty and readability are not proxies.

C. Infrequency of Zero Correlation

There is scant evidence of zero correlations (i.e., cells boxed as □) in Tables 5-7. There are some correlations that are near zero but they are not statistically significant.

D. NOI as a Contrary Indicator

Although categorically NOI is a debeautifying regime, statistically it behaves like a beautifying regime. This offers some insight into how beauty factors work. When a decrease in readability is accompanied by a increase in surface texture (i.e., B>0), there is negative correlation between beauty and readability. However, NOI decreases the average surface texture since more text is collected in the left edge of the artefact. This results in a positive correlation between de-beauty and readability. Generally, the debeautifying regimes in this paper tend to decrease surface texture while the beautifying regimes tend to increase surfacr texture.

E. REC as a Possible Contrary Indicator

In the cases, of Halstead statistics, REC appears to be a reasonable candidate as a contrary indicator. The correlation coefficient has the opposite direction implied by its modality for all readability scores. However, for PHD and SRES, the correlations are not statistically significant.

F. R2 and R5

Unlike NOI and to a less extent REC, the correlations for R2 and R5 are in the appropriate directions for their implied modality. R2 is not statistically significant for just one of the readability models (i.e., SRES) while R5 is not statistically significant for three of them (i.e., BW, PHD and SRES).

G. A case for "self-documenting" Code

Mnemonics are a form of documentation. The data for MNE suggests perhaps a more reliable approach to comment (i.e., as opposed to REC) to improve style is through mnemonics. That is make the code more "self-documenting" by choosing symbol names that reflect their use and aids in memory. While these data are not in any way intended to settle controversies concerning comments in code [43-45], it lends support to the "no comments" school, at least as far as aesthetics are concerned.

H. Improving Readability Through White Space

Tables 5-7 shows that all beauty treatments that affect basic tenet #1 (i.e., GNU, K&R, BSD, AND LIN) correspond consistently to improvements in readability. These treatments are more strongly correlated with readability than MNE or REC. In other words, it appears the most efficient means to improve readability through beautification is through basic tenet #1. NOI further supports this conclusion which is stronger than both NON, though the difference is not statistically significant, and DEC, where the difference is statistically significant.

I. GNU as a More Readable Style

Note further in Tables 5-7 that $|\rho|$ generally tends to be greater for GNU and the difference is statistically significant (P <0.05). The one exception is SRES versus GNU (ρ=-0.44) and BSD (ρ=-0.41): the pattern persists but it is not statistically significant (P=0.40). In other words, GNU tends to be more readable. Although the corpus is a GNU project, presumably written to the GNU standard, that fact should in theory should reduce the correlation because more files will have B=0. However, this is not the case which argues GNU, at least from a readability perspective, is a better style if we just go by the data.

VII. Conclusions

The data suggests that while some variations in beauty can be explained by variations in readability, most cannot be explained as such, at least not on the corpus in this study. In other words, beauty and readability are related as we hypothesized and beauty appears to measure a unique property in code called aesthetic appeal. The data further suggests that indentation is reliably correlated with readable code, more than mnemonics or comments and of the four styles, GNU style is the most correlated with readability. Future research needs to confirm these findings for different repositories, different languages and different styles. We believe this is a worthwhile endeavor with potential to inform certain deeply felt and passionately argued beliefs about style.

Acknowledgment

The author thanks Brendon Boldt for assistance with the ANTLR specification and Maria Luisa Coleman for reading drafts.

References

[1] D. E. Knuth, "Computer Programming as An Art," *CACM*, 17 (12), 1974

[2] Bell Telephone Laboratories, *Unix Seventh Edition Manual*, volume 1, 1979, http://plan9.bell-labs.com/7thEdMan/, last accessed: 10 Jul 2018

[3] B. Kernighan, D. Ritchie, *The C Programming Language*, Prentice Hall, 1978.

[4] R. Green and H. Ledgard, "Coding Guidelines: Finding the Art in the Science," *Communications of the ACM*, vol 52, issue 2, December 2011, p57-63, doi :10.1145/2043174.2043191

[5] M. J. Black, "The Art of Code," Ph.D. thesis, University of Pennsylvania, 1 Jan 2002

[6] D. Boswell and T. Foucher, *The Art of Readable Code*, O'Reilly, 2011

[7] H. Abelson and G. Sussman, *The Structure and Interpretation of Computer Programs*, 2nd ed., MIT Press, 1996

[8] P. Kokol, J. Brest, and V. Zumer, "Long-range correlations in computer programs," *Cybernetics and systems*, 28 (1), 43-57, 1997

[9] P. Kokol, and J. Brest, "Fractal structure of random programs," *SIGPLAN notices*, 33 (6), 1998

[10] P. Kokol, V. Podgorelec, and J. Brest, "A wishful complexity metric," in H. Combes (ed.), FESMA, p. 235-246, 1998

[11] R. Coleman, and P. Gandhi, "Fractal Analysis of Good Programming Style", *Proc. Second International Conference on Computer Science & Engineering*, Dubai, UAE, 28-29 Aug 2015

[12] H-O.Peltgen and P.H. Richter, P.H., *The Beauty of Fractals*, Springer, 1986

[13] R. Coleman and P. Gandhi, "Fractal Beauty of Programming Style," *Proc. of the 14th International Conference on Software Engineering Research and Practice*, SERP '16, CSREA Press, 2016

[14] R. Coleman and B. Boldt, "Aesthetics Versus Entropy in Source Code," *Proc. of the 15th International Conference on Software Engineering and Practice*, SERP '17, CSREA Press, 2017

[15] R. Coleman and M. Rahtelli, "A Fractal Geometry Approach to Quantifying Aesthetic Values in Scientific Codes", *Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence*, IEEE Computer Society, Editors: Hamid R. Arabnia, Leonidas Deligiannidis, Fernando G. Tinetti, Q-N. Tran, M. Qu Yang, ISBN-13: 978-1-5386-2652-8; BMS Part # CFP1771X-USB; DOI 10.1109/CSCI.2017.313

[16] R.P. Taylor, A.P. Micolich, and D. Jonas, "Fractal analysis of Pollock's drip paintings," *Nature* 399, 422 (3 June 1999), doi:10.1038/20833

[17] P. Gerl, Schönlieb, C., and K.C. Chieh Wang, "The Use of Fractal Dimension in Arts Analysis," *Harmonic and Fractal Image Analysis*, 2004, p70-73

[18] R.P. Taylor, R. Guzman, T.P. Martin, G.R.D. Hall, A.P. Micolich, D. Jonas, D., Scannell, M.S. Fairbanks, and C.A. Marlow, "Authenticating Pollock paintings using fractal geometry," *Pattern Recognition Letters*, Volume 28, Issue 6, 2007, p695-702

[19] J. Coddington, J. Elton, D. Rockmore, and Y. Wang, "Multifractal analysis and authentication of Jackson Pollock paintings," *Proc. Computer Image Analysis in the Study of Art* (SPIE 6810), 2008, doi: 10.1117/12.765015

[20] M. Irfan and D. Stork, "Multiple visual features for the computer authentication of Jackson Pollock's drip paintings: Beyond box counting and fractals," *Proc. SPIE 7251*, Image Processing: Machine Vision Applications II, 72510Q, 2009

[21] A. Oram and G. Wilson (eds.), *Beautiful Code: Leading Programmers Explain How They Think*, O'Reilly, 2007

[22] F. Brooks, "Towards a theory of the comprehension of computer programs," *International Journal of Man-Machine Studies*, 18, 543-554, 1984

[23] M.A., Storey, "Theories, tools, and research methods in program comprehension: past, present, and future," Software Quality J., 14:187-208, doi 10.1007/s11219-006-9216-4, 2006

[24] Stanford Encyclopedia of Philosophy, "The Concept of the Aesthetic," https://plato.stanford.edu/entries/aesthetic-concept/, last accessed 10 Jul 2018

[25] E. Avidan and D.G. Feitelson, "From obfuscation to comprehension," ACM SIGSOFT Software Engineering Notes, 23(3):75-77, 1998, Proceeding ICPC '15 Proceedings of the 2015, IEEE 23rd International Conference on Program Comprehension, pp. 178-181

[26] J.. Elshoff and M. Marcotty, "Improving Computer Program Readability to Aid Modification," *Communications of the ACM*, 25(8):512-521, 1982

[27] B. Mandelbrot, "How Long is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension," *Science*, 156 (3775)

[28] R. Stallman, *GNU Coding Standards*, Samurai Media Limited, 2015

[29] FreeBSD, "FreeBSD Kernel Developer's Manual," 2015 https://www.freebsd.org/cgi/man.cgi?query=style&sektion=9, last accessed: 10 Jul 2018

[30] L. Torvalds, "Linux Kernel Coding Style," http://slurm.schedmd.com/coding_style.pdf, last accessed: 10 Jul 2018

[31] J. Borstler, M.E. Caspersen, and M. Nordstrom, "Beauty and the Beast: Toward a Measurement Framework for Example Program Quality," *Software Quality Journal*, June 2006, Vol 24, Issue 2, pp. 231-246

[32] M. Halstead, *Elements of software science*, Elsevier, New York, 1977

[33] R.P.L. Buse and W. Weimer, "Learning a Metric for Code Readability," *IEEE Transactions on Software Engineering*, vol 4, issue 4, July 2010, p 546-558

[34] D. Posnett, A. Hindle, and P. Devanbu, "A Simpler Model of Software Readability," *Proceeding MSR '11 Proceedings of the 8th Working Conference on Mining Software Repositories*, Pages 73-82

[35] R. Flesch, "A New Readability Yardstick," *The Journal of Applied Psychology*, 32(3):221, 194

[36] A. Abbas, "Properties of Good Java Programs," Master's Thesis, Umea University, Sweden, 2009

[37] Cass, S. "The 2017 Top Popular Languages," *IEEE Spectrum*, 17 Jul 2017

[38] TIOBE: The Software Quality Company, "TIOBE Index for 2018," https://www.tiobe.com/tiobe-index/, last accessed: 11 Jul 2018

[39] C. Wood, J. Arceneaux, J. Kingdon, and D. Ingamells, "Indent," edition 2.2.10, for Indent Version 2.2.10, 23 July 2008, https://www.gnu.org/software/indent/manual/indent.pdf, retrieved 11 Jul 2018

[40] Free Software Foundation, Coreutils – GNU core utilities, http://www.gnu.org/software/coreutils/coreutils.html, last accessed: 11 Jul 2018

[41] D. Cornforth, H. Jelinek, and L. Peichl, "Fractop: A Tool for Automated Biological Image Classification," Proc. Sixth Australia-Japan Joint Workshop on Intelligent and Evolutionary Systems, 2002, p1-8

[42] R. Coleman, https://github.com/roncoleman125/Pretty, last accessed: 11 Jul 2018

[43] J. Kunk, "To Comment or Not Comment," *Visual Studio Magazine*, 1 June 2011, https://visualstudiomagazine.com/articles/2011/01/06/to-comment-or-not-to-comment.aspx, last accessed: 13 June 2017

[44] J. Raskin, "Comments are more important than code," *ACM Queue*, vol 3, issue 2, 18 Mar 2005

[45] P. Vogel, "Why You Shouldn't Comment (or Document) Code," *Visual Studio Magazine*, 27 June 2013, https://visualstudiomagazine.com/articles/2013/06/01/roc-rocks.aspx, last accessed: 13 June 2017