Aesthetics Versus Entropy in Source Code

Ron Coleman and Brendon Boldt
Computer Science Department, Marist College, Poughkeepsie, NY, USA

Abstract – *Although the separate literatures on programming style and information theories of software are large, little attention has been given to information theories of style. What is new in this paper we study the measure of programming style or "beauty" and its empirical relationship to disorder or information entropy in source. In a series of experiments, we use a beauty model based on fractal geometry and a corpus of programs from the GNU/Linux repository. The data show with statistical significance that beauty and entropy in code are inversely related. The data also indicate beauty and entropy are weakly to moderately correlated, which is to say, the beauty and entropy in source are not proxies. Finally, the data contains as a result of this study statistical evidence of ways in which the beauty model might serve as a new kind style checker. The main research contribution of this effort is a better understanding through empiricism of roles aesthetic expression and entropy play in crafting and maintaining codes.*

Keywords: programming style, fractal geometry, aesthetics, information entropy

1 Introduction

Software engineers have observed that a repository, during its lifecycle, often undergoes modifications that disrupt the layout or structure of the code—specifically, it's programming style—with changes believed to increase disorder or information entropy in the code [1, 2]. However, these observations remain anecdotal. Coleman and Gandhi [3] put forward a relativistic, fractal model of aesthetic appeal in code that makes two predictions related to crafting and maintaining codes. First, the measure of style or "beauty" as we define it here is inversely related to entropy in code. In other words, transformations that beautify code tend to have a lower bit rate and increased transparency of style while transformations that obfuscate or de-beautify code tend to have a higher bit rate and reduced transparency of style. The model furthermore predicts beauty and entropy are not proxies. That is, the correlation between beauty and entropy is weak-to-moderate.

What is new in this paper is we test these predictions through statistical experiments using the beauty model and semantic-preserving transformations on a corpus of open sources from the GNU/Linux source repository. The general goals of this research effort are to revisit the original ideas of Dijkstra [4], Kunth [5], and others regarding sensorial-emotional or aesthetic appeal in code [6]. However, our approach employs empirical methods which would enable programmers, educators, students and others to reason more objectively and systematically about programming style using metrics rather than anecdote, ad hoc assessments, guesswork, etc. Furthermore, in view of recent calls by industry and the government to expand computing instruction for kids, underrepresented groups, and the next generation of coders [7, 8], we posit that an understanding of styles and anti-styles through quantitative means may be more important than in the past.

2 Related work

Although the separate literatures on programming style and information theories of software are large (see for example [9, 10, 11, 12, 13, 14, 15, 16, 17]), little attention has been given to information theories of style. Kirk and Jenkins [18], Anckaert, et al [19], Moshen and Pinto [20], Moshen and Pinto [21], and Avidan and Feitelson [22] proposed to use information approaches to obfuscate code. However, they were fundamentally interested in information security, not the software development lifecycle. While Posnett, Hindle, and Devanbu [23] model incorporated information entropy in their logit regression model of readability, they were not investigating style but a means to simplify a readability model put forward by Weimer and Buse [24]. In both cases, they were interested not in aesthetics but readability. Kokol with others [25, 26, 27, 28] showed that programs indeed contained long-range correlations, i.e., code is self-similar, in lexical characters and tokens. However, they were not studying style but searching for a fractal-based metric of software complexity using generated Pascal programs. Coleman and Gandhi [35] showed that software complexity and beauty are related and in common sense directions with weak-to-moderate correlation. In other words, complexity and beauty are not proxies. The investigations of this paper resemble efforts of researchers who used fractal geometry to assess aesthetic values in paintings and masterpieces, including Pollock's "action paintings." [29, 30, 31, 32, 33] The main technical differences are the use of code versus fine art and programming style versus artistic style. *Beautiful Code* [34] deals with conceptual beauty in the design and analysis of algorithms, testing, and debugging, topics which are outside the scope of this paper.

3 Methods

In this section we give our methods, starting with some working definitions.

3.1 Some working definitions

We follow Coleman and Gandhi [3] who defined "style" operationally to be the layout of code, namely, its lexical

structure. They limited the scope to "basic tenets" of good programming style defined by three general recommendations from a survey of different style guides, namely: 1) use white space; 2) choose mnemonic names; and 3) include documentation. They then defined "beauty" to be the measure of style relative to the basic tenets.

3.2 Semantic-preserving transformations

Let S be some source code called the control or baseline. Then, we have

$$S' = T(S) \quad (1)$$

such that T is a transformation operator or "treatment" of S which results in a new source, S'. The sources, S and S', differ only in style; the semantics of S and S' are the same. There are two modalities of T: de-beautification and re-beautification. De-beautification manipulates S in ways inconsistent with the basic tenets. Beautification manipulates S ways consistent with the basic tenets. The tables below give the de-beautification and beautification treatments. (For details on the algorithms to manipulate the sources, see Coleman and Gandhi [35].)

Table 1 De-beautification treatments

T	Tenet	Semantics
NOI	1	Removes indents.
R2	1	Randomizes indents with 1-2 spaces.
R5	1	Randomizes indents with 1-5 spaces.
NON	2	Refactors names to be less mnemonic.
DEC	3	Removes comments.

Table 2 Beautification treatments

T	Tenet	Semantics
GNU	1	Applies GNU style [36].
K&R	1	Applies K&R style [37].
BSD	1	Applies BSD style [38].
LIN	1	Applies Linux style [39].
MNE	2	Refactors names to be more mnemonic.
REC	3	Adds comments.

3.3 Information entropy

If S = $\{s_0, s_1, s_2, \ldots\}$ where s_i are lexical tokens in S, then we compute, I, the information entropy (or bit rate) as

$$I(S) = -\sum_{i=1}^{n} p_i \log p_i \quad (2)$$

where p_i is the observed frequency of token, s_i [40].

3.4 Fractal dimension

Mandelbrot [41] described fractals as geometric objects that are self-similar on different scales and nowhere differentiable. A quantitative interpretation based on reticular cell counting or the box counting dimension, namely, gives D as

$$D(S) = \lim_{r \to 0} \frac{\log N_r(S)}{\log 1/r} \quad (3)$$

where S is some surface; $N_r(S)$ is the number of components or subcomponents covered by the ruler of size, r. For fractal objects, $\log N_r(S)$ will be greater than $\log 1/r$ by a fractional amount. If the ruler is a uniform grid of square cells, then a straight line passes through twice as many cells if the cell length is reduced by a factor of two. A fractal object passes through more than twice as many cells. For r, we use grid sizes of 2, 3, 4, 6, 8, 12, 16, 32, 64, and 128 measured in pixels [42].

3.5 Beauty model

Coleman and Gandhi [3] defined a beauty factor, B, to be

$$B(S|T) = k \log (D / D') \quad (4)$$

where k is a constant. When k=10 and the log is base 10, the units of B are decibels (dB). D and D' are the fractal dimensions of the sources, S and S', respectively, after they have been converted to "artefacts" which we describe further below. Beauty factor, B, has the following interpretations:

1. If B<0, the beauty of S might be improved given T.
2. If $B \geq 0$, the beauty of S probably won't be improved given T.

An artefact is an in-memory bitmap encoding or snapshot of a source. In effect, the measure of style by the model, B, is an image processing problem that measures the log difference in texture change. The literal artefact method (or LAM) encodes the bitmap using a fixed width font such that the representation visually looks like S or S' literally. However, 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. It leaves spaces as blanks. BAM offers some advantages over LAM, the primary one being BAM is more robust against language dependencies. Furthermore, BAM effectively destroys the source readability in favor of the text's visual-spatial pattern so as not to confound aesthetic appeal and readability.

3.6 Statistical methods

To measure the correlation, we use Spearman's rank correlation coefficient or rho (ρ). We use the standard correlation definitions of "none" as ρ=0; "weak" as $0<|\rho|\leq 0.30$; "moderate" as $0.30\leq|\rho|\leq 0.70$; and "strong" as $0.70\leq|\rho|\leq 1.00$. These classifications are guidelines only. Since there may be borderline situations, the definitions are best interpreted with some flexibility.

4 Experimental design

We use as the test bed code from the GNU Core Utilities which have been decomposed into 1,043 single-function C files or 57 KLOC. These files have been stripped of compiler and preprocessor directives, prototype declarations, typedefs, and most comments external to the function. Only comments perceived to be related to a function remain. We measure the entropy, *I*, for each of these files. We similarly invoke the 11 treatments in Tables 1 and 2 on these files, convert S and S' to artefacts, measure the fractal dimensions, and finally calculate B for each file. As we indicated above, we convert these measures to ranks and then measure the corresponding correlation coefficient on I and B.

5 Results

Results of the experiments is in the table below.

Table 3. Correlation coefficients of *I* v. *B*(*S*|*T*)

Modality	T	ρ	*P*
De-beautify	NOI	-0.51	$<10^{-4}$
	R2	0.13	$<10^{-4}$
	R5	0.07	0.02
	NON	0.46	$<10^{-4}$
	DEC	0.28	$<10^{-4}$
Beautify	GNU	-0.55	$<10^{-4}$
	K&R	-0.43	$<10^{-4}$
	BSD	-0.35	$<10^{-4}$
	LIN	-0.50	$<10^{-4}$
	MNE	-0.25	$<10^{-4}$
	REC	0.07	0.01

5.1 De-beautify distributions

Results of de-beautifications are in the scatterplots below.

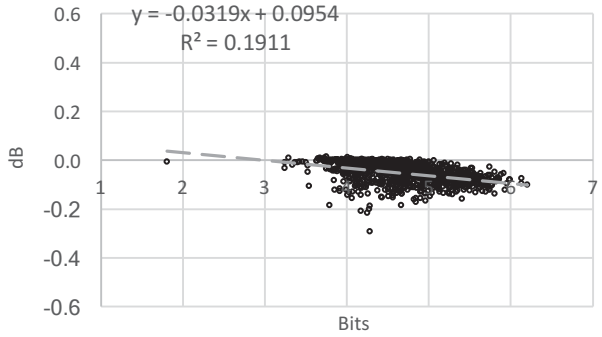

Figure 1. I vs. NOI

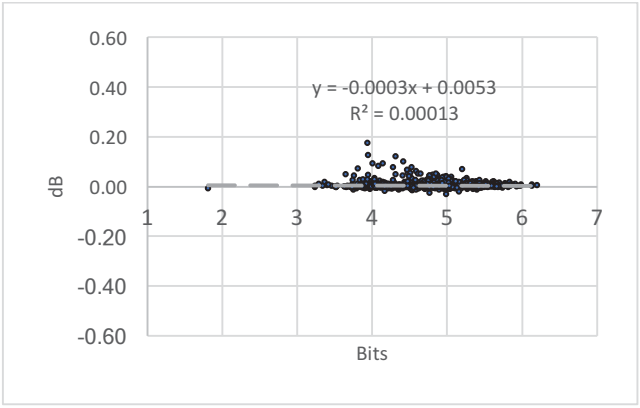

Figure 2. I vs. R2

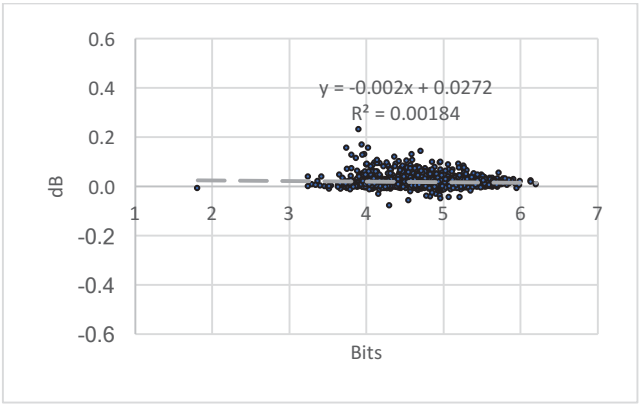

Figure 3. I vs. R5

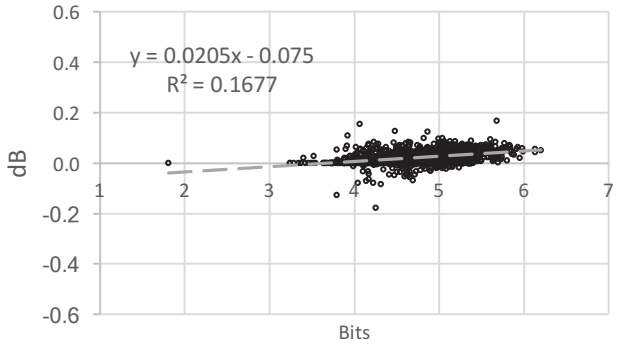

Figure 4. I vs. NON

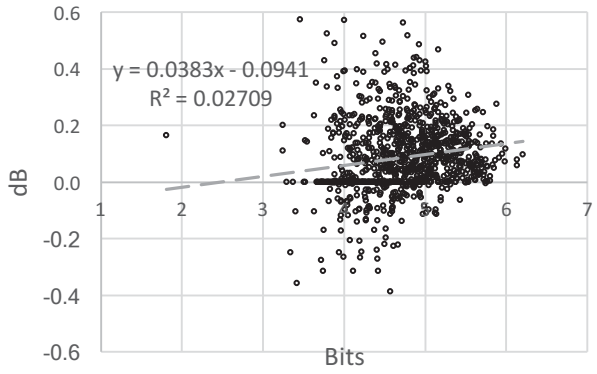

Figure 5. I vs. DEC

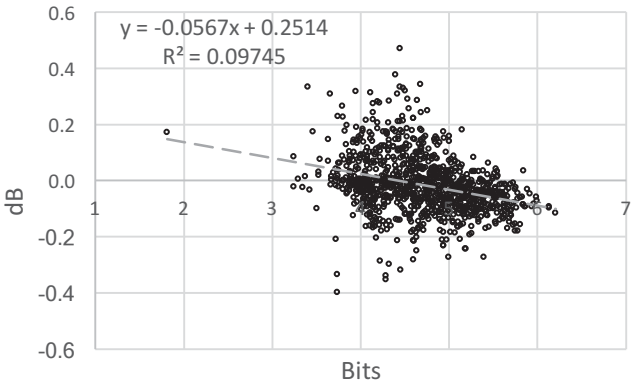

Figure 8. I vs. BSD

5.2 Beautify distributions

Results of beautifications are in the scatterplots below.

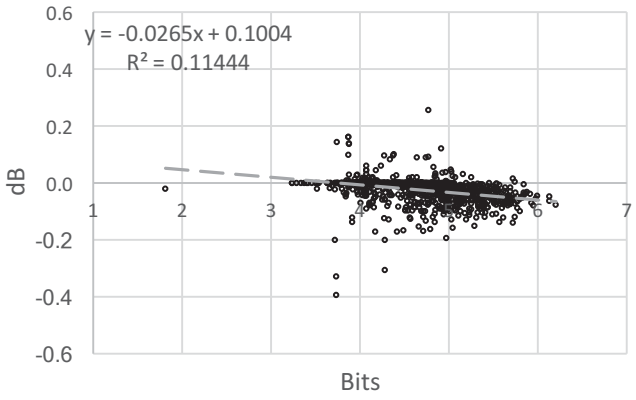

Figure 6. I vs. GNU

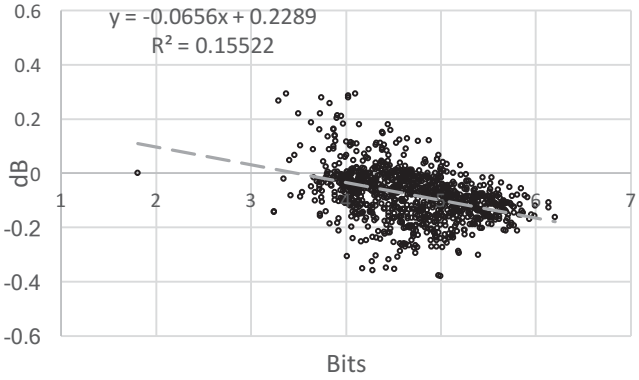

Figure 7. I vs. K&R

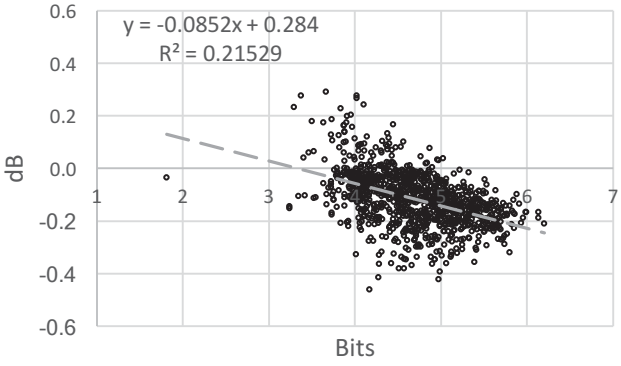

Figure 9. I vs. LIN

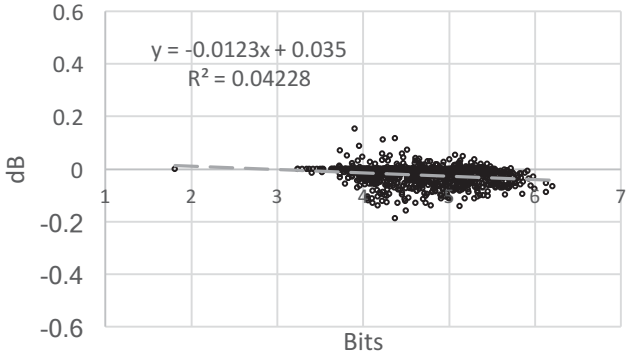

Figure 10. I vs. MNE

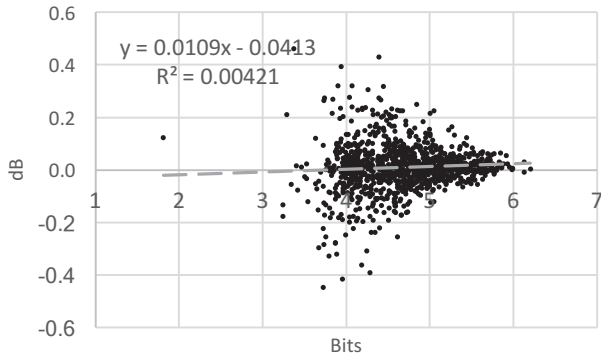

Figure 11. I vs. REC

6 Discussion

The data in Table 2 generally support model predictions that beauty and entropy are inversely related and furthermore, the correlations are only weak-to-moderate. While these patterns are more or less clear for both de-beautification and re-beautification treatments, the case for re-beautification appears to be stronger and more consistent. However, we argue this is to be expected and serves as a kind of "sanity test." That is, the de-beautification regimes depend more randomization, namely, R2 and R5, for which by definition there would be weaker correlation. In fact, REC which adds comments randomly from a database of comments is similar: weak correlation and relatively less statistical significance. Indeed, the stronger correlations and greater statistical significances are in those treatments (see below) that are non-stochastic in nature.

NOI is negatively correlated with de-beautification treatment. In other words, it as a reliable yet "contrary indicator." Conceivably in practical terms, it suggests maintaining code with less attention given to indentation and possibly to the extreme of eliminating it, increases the code entropy. NON and DEC are both moderately correlated with entropy and statistically significant. Removing mnemonics and comments are consistent with fixing code with less care given to mnemonics and maintaining the documentation. Each of the other beautifying treatments, GNU, K&R, BSD, LIN, and MNE, improve style and reduce entropy with correlation that is generally moderate with statistical significance.

Figures 3-11 support these conclusions. As the figures suggest, the median bit rate in the test bed is between four and five bits with IRQ/median = 0.19. The median beauty varies between -0.11 dB (LIN) and 0.06 dB (DEC). However, beauty |IRQ/median| ratio varies from minimum of 1.16 (LIN) to 22.7 (R2). We conclude from this data that beauty is a more sensitive measure by at least nearly tenfold compared to entropy. In other words, the general shape or envelop of the distributions are the same but the slope and dispersion are not the same by different degrees. We note furthermore there are clusters of data points with B=0 dB. The table below gives the frequency counts and percentages of the complete corpus where this occurs.

Table 4 Frequency and percentage in which B=0

T	Freq.	%
NOI	12	1
R2	12	1
R5	4	0
NON	189	18
DEC	218	21
GNU	237	23
K&R	0	0
BSD	0	0
LIN	0	0
MNE	256	25
REC	0	0

Recall B=0 when D=D' which occurs when a treatment does not affect the style. This could occur for a variety of reasons that depend on T. It is indicative of how B can also work as a new kind of style checker. For instance, for NOI the data suggests 99% of the files use some sort of indentation. According to DEC, 21% of the files have no comments. Only 23% of the time programmers use GNU style, although the corpus is from a GNU project. While NON and MNE individually offer insight into how names are being used stylistically, comparing them would appear to offer additional insight. Namely, in the interests of good programming style we would expect ||NON|| > ||MNE|| where ||•|| denotes the frequency where B=0. This relation implies there are more opportunities to make the code less mnemonic than more mnemonic, i.e., there are more high frequency symbols to shorten than there are to lengthen. However, the data in Table 4 shows ||NON|| < ||MNE||, the difference of which is statistically significant ($P<10^{-3}$, two tailed). In other words, in this code base, there appears to be relatively more opportunities to make the code more mnemonic.

7 Conclusions

We have shown that the data generally supports the two predictions we set out to investigate, namely, beauty and entropy are inversely related and they are not proxies. The data furthermore suggests how the beauty model could also serve as a style checker. Future research needs investigate whether patterns this study has identified are persist across differ code bases and whether the patterns are stable over time or evolve. One could conceivably do what amounts to a longitudinal study using different releases and directly test the hypothesis of increasing entropy and its relationship to beauty.

8 References

[1] Lehman, M., Belady, L., Program evolution: processes of software change, Academic Press Professional, Inc., 1985, San Diego, CA

[2] Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, Gunnar, Object-Oriented Software Engineering: A Use Case Driven Approach, ACM Press. Addison–Wesley, 1992, pp. 69–70, ISBN 0-201-54435-0

[3] Coleman, R., and Gandhi, P., "Fractal Beauty of Programming Style," Proc. 14th International Conference on Software Engineering, Las Vegas, NV, 25-28 Jul, 2016

[4] Dijkstra, E., "A short introduction to the art of programming," August 1971, EWD316, Edsger W. Dijkstra Archive, University of Texas, http://www.cs.utexas.edu/~EWD/welcome.html, last accessed: 9 January 2017

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

[6] Martin, R.C., *Clean Code*, Prentice Hall, 2008

[7] Paul, A. M., "The Coding Revolution," *Scientific American*, August 2016

[8] Margolis, J., *Stuck in the Shallow End: Education, Race, and Computing*, MIT Press, 2008

[9] Berry, R.E., and Meekings, B.A.E., "A style analysis of C programs," CACM, Vol 28, Issue1, Jan 1985

[10] Redish, W.F., and Smyth, W.F., "Program style analysis: a natural product of program compilations," CACM, Volume 29, Issue 2, Feb 1986

[11] Oman, P.W., and Cook, C.R., "A paradigm for programming style research," ACM SIGPLAN Notices, Vol 23, Issue 12, Dec. 1988, pages 69-78

[12] Oman, P.W., Cook, C.R., "A taxonomy of programming style," CSC '90 Proceedings of the 1990 ACM annual conference on Cooperation, Washington, DC, 20-22 Feb, 1990, ACM, 1990

[13] Khoshgoftaar, T.M., and Allen, E.B., "Applications of information theory to software engineering measurement," Vol 3, Issue 2, Software Quality Journal, June 1994, pp79-103

[14] Allen, E.B., "Measuring Coupling and Cohesion: An Information-Theory Approach," Proceedings Sixth International Software Metrics Symposium, 1999, IEEE, 10.1109/METRIC.1999.809733

[15] Allen, E.B., and Khoshgoftaar, T.M., "Measuring coupling and cohesion of software modules: an information-theory approach," Proc. Seventh International Software Metrics Symposium, 2001, IEEE, METRICS 2001, 0.1109/METRIC.2001.915521

[16] Khoshgoftaar, T.M., "Measuring coupling and cohesion of software modules: an information-theory approach," Proc. Seventh International Software Metrics Symposium, 2001, IEEE, METRICS 2001, 0.1109/METRIC.2001.915521

[17] Bailetti, A.J., Liu, J., "Comparing software development processes using information theory," PICMET '03 Portland International Conference on Management of Engineering and Technology, 24 Jul, 2003, IEEE, 10.1109/PICMET.2003.1222808

[18] Kirk, S.R., and Jenkins, S., "Information theory-based software metrics and obfuscation," *Journal of Systems and Software*, 72(2):179-186 · July 2004

[19] Anckaert, B., Madou, M., De Sutter, De Bus, B., and De Bosschere, K., "Program Obfuscation: Quantitative Approach," QoP '07 Proceedings of the 2007 ACM workshop on Quality of protection

[20] Moshen, R. and Pinto, A.M., "Algorithmic Information Theory for Obfuscation Security," 12th International Joint Conference on e-Business and Telecommunications (ICETE), 2015a

[21] Moshen, R. and Pinto, A.M., "Evaluating Obfuscation Security: A Quantitative Approach," 8th International Symposium on Foundations & Practice of Security-fps2015, at Clermont-Ferrand, France, 2015b

[22] Avidan, E., and Feitelson, D. G., "From obfuscation to comprehension," ICPC '15 Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, Florence, Italy, 2015

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

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

[25] Kokol, P., "Searching for fractal structure in computer programs," SIGPLAN notices, 29 (1), 1999

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

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

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

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

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

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

[32] Coddington, J., Elton, J., Rockmore, D., and Wang, Y., "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

[33] Irfan, M., and Stork, D., "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

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

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

[36] FreeBSD. "FreeBSD Kernel Developer's Manual," https://www.freebsd.org/cgi/man.cgi?query=style&sektion=9, last accessed: 28 April 2017

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

[38] FreeBSD. "FreeBSD Kernel Developer's Manual," https://www.freebsd.org/cgi/man.cgi?query=style&sektion=9, last accessed: 28 April 2017

[39] Torvalds, L., "Linux Kernel Coding Style," http://slurm.schedmd.com/coding_style.pdf, last access: 28 April 2017

[40] Pierce, J.R., An Introduction to Information Theory, Dover, 1980

[41] Mandelbrot, B., "How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension", Science, 156 (3775): 636–638, 1967, doi:10.1126/science.156.3775.636.

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