ISSN: 2473-4901
v11 n6333

An Analysis of Natural Language and Java Program Complexity

Jon D. Clark
jon.clark@colostate.edu

Seth J. Kinnett
seth.kinnett@colostate.edu

Computer Information Systems, College of Business
Colorado State University
Fort Collins, CO 80523, USA

Abstract

The purpose of this paper is to explore the correspondence between student programmer Natural Language and Program Complexity in an educational project setting. Samples of text written by each of the members of a Java programming class were compared with software complexity metrics. The written text was evaluated with the Flesch Reading-Ease, the Flesch-Kinkaid Grade Level and Gunning Fog indexes, and the Java programs were scanned for McCabe metrics for both level of expression and complexity. The McCabe metrics of v(G) Cyclomatic Complexity and ev(G) Essential Complexity were used to categorize submitted and working programs into the Unmaintainable and Maintainable groups. The metrics for each student's text and programs were then compared. The idea that started the research was born during the presentation of *A Study of Software Metrics, Student Learning, and System Development Metrics*, at the 2024 ISCAP Conference. At a deeper level, the question of whether expressions of the written word as well as programs share fundamental cognitive process and outcome and, therefore, might be used as a predictive tool for programmer performance.

Keywords: Natural Language Metrics, Software Metrics, Halstead, McCabe, Vibe Coding

ISSN: 2473-4901
v11 n6333

An Analysis of Natural Language and Java Program Complexity

Jon D. Clark and Seth J. Kinnett

1.INTRODUCTION

Improving student outcomes in computer programming courses is a priority for Information Systems instructors. Although remedial approaches can be implemented upon identification of particular student struggles, such approaches rely on some critical mass of assignments to be completed, so instructors can determine topics of concern for particular students. This challenge is compounded by the reality that introductory programming courses may not cover a substantive number of topics until several weeks into a given term, not to mention resource and bandwidth limitations could impact instructors' abilities to engage in remedial techniques. One ideal intervention centers upon the potential to identify at the very beginning of the course those students that might be expected to struggle. Implementing a programming assessment would be ineffective since all students are presumed to be beginners, resulting in the need for some form of non-programming assessment.

This paper provides insights into the possible relationship between one's complexity and level of written natural language expression and the corresponding complexity and level of expression in a programming language. This work extends what was done by Kinnett and Clark (2024) and presented at the ISCAP Conference that year. That research was directed at the relationship between software metrics (Halstead, 1977 and McCabe, 1976, 2024) and student learning. Findings included nuances about grading rubrics as well as general mutual support between Halstead and McCabe metrics. This was of interest since the two sets of metrics are based on quite different bases: information theory and program control paths. Due to a comment provided by an attendee at the presentation and a short conversation we decided to explore whether there is a relationship between natural language expression and that of computer programming. If such a relationship exists, then there is a possible shared cognitive process that is used and possible related outcomes that are quite important and possibly useful.

There are 29 Java programs used in this study, all of which satisfy a single in-class assignment in CIS 240, Application Design and Development. This is the first programming course that Computer Information Systems majors take. The 16-week course introduces students to object-oriented programming fundamentals using Java, spanning the concepts outlined in Table 1. In addition, an assignment in CIS360, Systems Analysis and Design, involved English text, and was used to derive several linguistic metrics. These metrics were used as a predictor of program complexity in the McCabe groups of Unmaintainable and Maintainable obtained in the CIS240 programming course. It is important to note that CIS360 concentrates on the Universal Modeling Language (UML) and not on computer programming.

Module Number	Topics
1	Course Overview & History of Programming Languages
2	Java Fundamentals (data typing, variables, constants)
3	Selection Statements (if/else/else if/switch)
4	Loops (while, do-while, for)
5	Methods & Method Overloading
6	Arrays (one and two-dimensional)
7	Classes & Objects (data encapsulation, constructors)
8	String Manipulation & File I/O

Table 1: Course Topic Summary

Natural Language Metrics

Language complexity can be captured by a number of characteristics relative to structure. With approximately 6,000 natural languages in existence, there are many variations in syntax and complexity. According to Rescher (1998) and Sinnemaki (2011) the general categories of complexity can be decomposed into the following:

- Syntagmatic complexity: number of parts, such as word length in terms of phonemes, syllables etc.
- Paradigmatic complexity: variety of parts, such as phoneme inventory size, number of distinctions in a grammatical category or aspect.
- Organizational complexity: ways of arranging components, phonotactic restrictions, and variety of word orders.
- Hierarchic complexity: such as recursion, and lexical-semantic hierarchies.

ISSN: 2473-4901
v11 n6333

To capture useful and easily obtainable Natural Language (NL) metrics for this research, a variety of formulas were considered. Fortunately, beginning in the 1930's following the Great Depression, a number of readability formulas were developed. These focus on readability relative to the level of education. Rudolf Flesch (1955) produced the Flesch Reading Ease formula as a tool to enhance marketing of print matter. By 1975, the Flesch-Kincaid (Kincaid, 1988) partnership with the US Navy was formed. This metric became known as the Flesch-Kincaid Grade Level index based on the US educational system. In addition, Robert Gunning produced a formula which became known as the Gunning Fog index around 1952, with some changes in the formula that took place over the next several decades. The purpose of the index was to measure the degree of understandability of text in the newspaper and textbook publishing domain.

The metrics chosen for this study are commonly accepted, used, and serve as the independent variable: 1) Flesch Reading Ease; 2) Flesch-Kincaid Grade Level; and 3) Gunning Fog Index. These metrics address the readability of text based on measurable characteristics of a sample text for the purpose of assessing the ease with which a reader can consume and understand the passage. The Flesch Reading Ease metric is based on a similar formula with different weights and an index that begins at 0.0 through 100.0 and is a reciprocal of the Gunning Fog Index. A score of 100.0 is associated with 5th grade and 0.0 with college educated professionals. The formula is as follows:

Flesch Reading Ease = 206.835 – 1.015(total words/total sentences) – 84.6 (total syllables/total words)

This index applied to a sample of *Reader's Digest* has an index of 65 (between 8th and 9th grade), *Harvard Law Review* in the low 30s. Florida insurance policies have an index of 45 or greater (some college).

The Flesch-Kincaid Grade Level metric is based on the following formula:

Flesch-Kincaid Grade Level = 0.39 (Total Words/Total Sentences) + 11.8 (Total Syllables/Total Words) – 15.59

In the case of the Gunning Fog Index, the formula is as follows:

Fog Index = 0.4 (words/sentences) + 100 (complex words/words)

Where complex words consist of three or more syllables do not include proper nouns, familiar jargon, or compound words. Neither are common suffixes such as -es, -ed, and -ing counted as syllables.

It's well recognized that this index has its limitations, was intended for English, and may not be correct for other languages. The range of values produced range typically from 4 (fourth grade, through 17 (college graduate).

According to Gunning (p. 4-5,1969):

> "In 1944 I setup Robert Gunning Associates to help staffs of publications and corporations improve their writing. The Gunning Fog Index resulted from our efforts to produce a measure that would be sufficiently reliable and still easy to use. Apparently, this effort has been helpful to a great many people. The Army, Navy and Air Force chose this formula for their writing manuals, and we have given them permission to use it."

Not incidentally, the Gunning Fog Index is generally available and can be used on text produced in MS Word.

McCabe Control Flow Metrics
McCabe's complexity measures were based on graphs of control flow, where nodes represent program statements and edges (arcs) represent the flow. Statements that determine decisions produce branches in the graphs and the count of various paths are an important determinant of complexity. These metrics are far more domain specific to procedural programming than Halstead's approach but are not predictive of effort across the stages of systems development.

The control graph produces the following metrics:

E = number edges of the graph

N = number of nodes of the graph

P = number of connected components (program exit points)

The derived metrics are as follows:

v(G) (Cyclomatic Complexity) = E-N+2: number of edges less the number of nodes plus the number of connected components

ev(G) (Essential Complexity) = 1<= ev(G) <= v(G): based on reduced control flow graph

Interpretation of v(G) thresholds by McCabe:

- 1-10: simple procedure, little risk
- 11-20: more complex, moderate risk
- 21-50: complex, high risk
- >50: untestable code, very high risk

The Essential Complexity, ev(G) is produced by removing all the structured programming primitives. These include 1) sequence; 2) selection statements, including *if* and *case* statements; 3) iteration constructs, including *while, do,* and *for*.

2. RESEARCH DESIGN & QUESTIONS

The dataset to be used in this research consists of a sample of 29 successful attempts at producing a solution to Java programming assignment for the Rock, Paper, and Scissors game. This assignment is one of many, approximately halfway through a 16-week semester and is denoted as ICE04 (In Class Exercise 04) and features concepts related to loops. These exercises are carefully controlled in a classroom setting for a period of 75 minutes of individual programming effort. See **Figure 1: ICE04 Assignment** and **Table 2: ICE04 Grading Rubric**.

Write a Java program that plays the game Rock, Paper, Scissors.
The rules are as follows:
•Rock (0) beats scissors (2)
•Scissors (2) beats paper (1)
•Paper (1) beats rock (0)

At the start of the program, the program must ask the user for their name. The program should then ask how many rounds the player wants to play. The program will then prompt the user to choose rock, paper, or scissors and randomly choose a value for the computer player. It will determine the winner of that hand, display the results, and keep track of the number of hands won by the player, the number won by the computer, and the number of tie games.

Use JOptionPane for all inputs and outputs.

Figure 1: ICE04 Assignment

Proper coding habits including indentation & comments (1)
Proper compilation (no errors) (2)
Either *for* or *while* loop implemented properly to loop for the number of rounds specified by the user (2)
Correct use of if/else-if or switch to process user's choice each time (1)
Correct use of nested if/else-if to evaluate computer's choice (1)
Correctly implements counter variables to track computer wins, player wins, ties (1)
Correct computation of winner using if/else-if/else to evaluate the counters (1)
Correct generation of output using string concatenation (1)

Table 2: ICE04 Grading Rubric

Each program submitted and evaluated as successful according to the rubric was analyzed by *BattleMap IQ*, a tool that produces McCabe metrics from source code, in this case from Java. The **APPENDIX** has a table of values obtained in this manner. In addition, each student represented in this table was also evaluated in terms of a writing exercise from a class assignment in which English text was required and each such sample of writing was analyzed using the document evaluation tool contained in MS Word. The dataset consists of McCabe's Cyclometric measure v(G) and Essential Complexity ev(G), as well as Flesch Reading Ease, Flesch-Kincaid Grade Level, and Gunning Fog index.

It should be noted that the table contained in the **APPENDIX** has been partitioned into the McCabe categories of *Maintainable* and *Unmaintainable.* These are highly significant and based on a threshold where of 4 for Essential Complexity where if ev(G) > 4 the code is *Unmaintainable* for ev(G) =< 4 is *Maintainable*. This threshold has been determined by McCabe based on experience. Additionally, a threshold of 10 has been used by McCabe with regard to Cyclomatic Complexity where v(G) =< 10 is R*eliable* and v(G) > 10 is *Unreliable*. Surprisingly, all the student programs fell into the category of *Unreliable* in either *Maintainable* or *Unmaintainable!* See Figure 2 for the scatterplot of unmaintainable and maintainable complete working programs.

The of distinction between Reliability and Maintainability may appear to be confusing. According to McCabe, the two subcategories of *Reliability* and *Unreliability* is based on v(G), a measure of decision structure complexity. If v(G) is greater than 10 then the module would be difficult to test. In a similar fashion, the

ISSN: 2473-4901
v11 n6333

subcategories of *Maintainable* and *Unmaintainable* are based on ev(G), a measure of unstructuredness. If ev(G) is greater than 4 then the module makes use of unstructured programming constructs resulting in maintenance issues. A paper by Riabov (2007) provides an applied demonstration of these constructs.

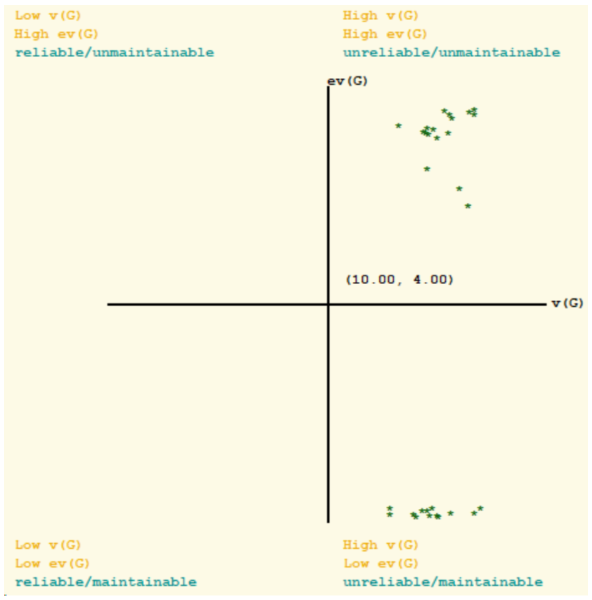

Figure 2: v(G) & ev(G) Scatterplot of

Student Submissions

The purpose of this research is to explore the correspondence between Natural Language and Program Complexity in an educational project setting. To carry out this evaluation, the dataset having fully functional programs that met the goals of the rubric, have been partitioned into *Unmaintainable* (UM) and *Maintainable* (M) categories. The following Research Questions (RQ) will be addressed:

RQ1: Is there a relationship between Flesch Reading Ease mean scores and UM and M categories determined by McCabe's ev(G)?

> ($H_0$1) There is no difference in mean Flesch Reading Ease scores between the UM and M categories determined by McCabe's ev(G)

> ($H_1$1) There is a difference in mean Flesch Reading Ease between the UM and M categories.

RQ2: Is there a relationship between Flesh-Kincaid Grave Level mean scores across the UM and M categories determined by McCabe's ev(G)?

> ($H_0$2) There is no difference in mean Flesch-Kincaid Grade Level scores across the UM and M categories determined by McCabe's ev(G)

> ($H_1$2) There is a difference in mean Flesch-Kincaid Grade Level scores between the UM and M categories determined by McCabe's ev(G)

RQ3: Is there a relationship between mean Gunning Fog Index scores across the UM and M categories determined by McCabe's ev(G)?

> ($H_0$3) There is no difference in mean Gunning Fog Index scores between the UM and M categories determined by McCabe's ev(G)

> ($H_1$3) There is a difference between mean Gunning Fog Index scores between the UM and M categories determined by McCabe's ev(G)

A bootstrapped independent samples t-test will be used to determine whether there is a relationship between English Natural Language Complexity and the Program Complexity produced. If there is a relationship, one might use this to better manage the programming process and produce better, perhaps more reliable and maintainable results.

3. RESULTS

An independent samples t-test evaluating mean differences across samples is an appropriate approach in this circumstance. As the gold standard for sample size in parametric statistical tests is a minimum of 30 observations per group in order to satisfy the Central Limit Theorem, we opted to employ bootstrapping upon our samples ($N_{maintainable} = 11$, $N_{unmaintainable} = 18$). Bootstrapping is a well-established and reliable technique to increase the reliability of findings when small sample sizes are present. Accordingly, we performed the difference of means independent samples t-test using 1000 simulated bootstrapped samples using SPSS to determine the degree of independence between the maintainable and unmaintainable categories of the samples. We utilized SPSS defaults for the technique, designating a simple (vs. stratified) sampling scheme and 90% confidence intervals. Interpreting the results required us to evaluate the results of Levene's test for equality of variance. In all three calculations, Levene's test indicated variance between groups was equal. Table 3 shows that both the Flesch-Kincaid Grade Level and the Gunning Fog Index are significant

ISSN: 2473-4901
v11 n6333

at the 0.09 level. In accordance with best practices surrounding the use of bootstrapping, we evaluate our decisions regarding rejection or failure to reject null hypotheses based on the bootstrapped confidence intervals. Intervals spanning zero indicate that one possible result is no mean difference, which would cause us to fail to reject null hypotheses. This method is preferred over simple interpretation of p-values. In our sample, we generally see alignment except with the Flesh index, which has confidence intervals that support rejecting the null hypothesis and p-values that suggest retaining it. Table 3 shows the summary findings from our bootstrapped independent samples t-test.

Metric	**MD**	**Bias**	σ_M	***p***	**CI$_{90}$**
Flesh Index	9.97	-.27	5.95	.12	[.04, 19.5]
Grade Level	-1.7	.03	.97	.098	[-3.2, -.10]
FOG Index	-1.8	.03	.98	.086	[-3.4, -.20]

Table 3: Bootstrap Independent Samples Test

RQ1 asks *Is there a relationship between the Flesch Reading Ease and McCabe metrics determining the* UM and M categories*?* Even though the p value with respect to the M and UM categories is just 0.122, the 90% confidence intervals do not span zero (CI$_{90}$ = [.0391, 19.524]), leading us to reject $H_0$1. The Flesch Reading Ease index is based on the number of words per sentence and the number of syllables per word. Programming constructs and control flow appear to be predictable based on natural language constructs.

RQ2 asks *Is there a relationship between Flesch-Kincaid Grade Level and McCabe metrics determining the UM and M categories?* In this case, the p value is 0.098 and CI$_{90}$ = [-3.159, -.0984]. The Flesch-Kincaid Grade Level, though based on the same variables, uses quite different weights and is the reciprocal of the Flesch Reading Ease Index. One might conclude that the grade level has a better fit as far as prediction of M and UM categories.

RQ3 asks *Is there a relationship between the Gunning Fog index and McCabe metrics determining the UM and M categories?* This case has a p value of 0.09 and CI90 = [-3.41, -.195]. The p value of the Gunning Fog Index is the best predictor of M and UM categories. Again, this index is based on weighted words per sentence and complex words of three or more syllables compared to the total number of words used.

As we can see, we found significant differences across all three natural language metrics when comparing the two groups of maintainable vs. unmaintainable code based on McCabe's ev(G).

4. DISCUSSION AND LIMITATIONS

These findings lead to a number of interpretations and pedagogical implications. First, the idea that one aspect of code quality can essentially be predicted by an English language writing sample is – to our knowledge – a novel finding. We suggest that instructors could administer a writing prompt at the start of the term for students enrolled in a programming course to get a sense of which students might need more assistance in writing quality code.

Student access to AI Has increased reliance on the use of available partial solutions as a development starting point. Vibe coding (Karpathy, 2025), a new type of process in which NL requests are made to a Large Language Model (LLM), has begun to receive attention. This process allows iterative refinement of a problem statement in which code is progressively refined as well. As an example, when a block of code is returned from a request, either functional extensions may be added or corrections to compile or runtime errors. Vibe coding has become an accepted practice by novice programmers in both academic and applied settings for simple application development. Andrew Ng (2025) is offering course material and training in this new process. Certainly, there are critics, but as AI develops, the effectiveness of the approach may well change. Given the relationship found in this paper, there is a possibility of further research between the NL component of the requests provided to the LLM and the degree of precision of the coded outcome. The research convergence of NL and coding deserves greater attention as shown in this research.

Focusing on the FOG index, given its higher apparent predictive power, we suspect the findings explain the idea that succinct English sentence construction correlates with fewer control flow paths (measured by ev(G)), suggesting higher cognitive organization, whereas higher FOG scores correspond with higher ev(G) values, suggesting a more meandering style of coding and writing English that both demonstrate less forethought.

We suggest that students are essentially less focused in their production of both English sentences and Java code. Such students need to plan their approach more in both arenas and likely are jumping right into both exercises with a *think as you go* approach, which leads to a decrease in both code structuredness and the more complicated sentence constructions.

Pedagogically, all students received the same instruction, yet some students wrote maintainable code "naturally" while others didn't.

5. CONCLUSIONS

Now that we have statistically established that there is a relationship between NL understandability and the McCabe categories of UM and M code, several other studies may determine the possibility and usefulness from a predictive point of view. A few student-centered possibilities are:

- Should students with low NL understandability be treated differently, and if so, how and when?
- Can NL training be used to advantage before or along with coding?
- Should students with low NL understandability be encouraged to choose another field?

Since it appears that students with more succinct writing styles also tend to write more understandable code, the above possibilities are of interest.

In addition, further research may be needed to determine the impact of pedagogical impact of coding practice. The UM category contains a great deal of unstructured code. A few pedagogical studies may include:

- Does a time constraint affect structure?
- Does coding environment play a role?
- What role does application complexity play in the structure of the solution?
- Will vibe coding mitigate the impact of NL understandability?

The connection between NL and code understandability encourages us to further study the development processes used and the pedagogical impact of teaching methods.

REFERENCES

Flesch, R. F. (1955). *Why Johnny can't read, and what you can do about it*. https://www.amazon.com/Why-Johnny-Cant-Read-about/dp/0060913401

Gunning, Robert (January 1, 1969), The Fog Index After Twenty Years, *Journal of Business Communication*, Volume 6 (2), https://doi.org/10.1177/002194366900600202

Halstead, Maurice H. (1977), Elements of Software Science, Elsevier. https://dl.acm.org/doi/10.5555/540137

Karpathy, Andrej (2025, February 2). Tweet archived from the original on April 17, 2025. Retrieved May 9, 2025 via Twitter.

Kincaid JP, Braby R, Mears J (1988). "Electronic authoring and delivery of technical information". Journal of Instructional Development. 11 (2): 8– 13. doi:10.1007/bf02904998. S2CID 62551107.

Kinnett, Seth J. & Clark Jon D. (2024, November 6-9), A Study of Software Metrics, Student Learning, and System Development Metrics, *Proceedings of ISCAP Conference,* Baltimore, Maryland.

Mccabe.com (2024, May 15). McCabe Software: The software Path Analysis Company. Retrieved from https://mccabe.com.

McCabe, Thomas J. (1976, December), A complexity measure, *IEEE Transactions on Software Engineering,* SE-2(4):308-320, https://doi.org/10.1109/TSE.1976.233837.

Ng, Andrew (2025, May 28). Blockchain.News, https://blockchain.news/flashnews/replitlaunches-vibe-coding-101-for-ai-driven-application-development

Rescher, Nicholas (1988). *Complexity: A Philosophical Overview.* New Brunswick: Transaction Publishers. ISBN 978-1560003779.

Riabov, Vladimir V. (2007). Graph Theory Applications in Developing Software Test Strategies for Networking Systems, *River Academic Journal*, 3(1), Spring.

Sinnemaki, Kaius (2011). *Language universals and linguistic complexity: Three case studies in core argument marking* (http://urn.fi/URN:ISBN:978-952-10-7259-

ISSN: 2473-4901
v11 n6333

8) (Thesis). University of Helsinki. Retrieved 2016-04-28.

ISSN: 2473-4901
v11 n6333

APPENDIX
Java Program Submissions with Metrics

ID	Group	v(G)	ev(G)	FlshRead	FlshKinGrade	SFOG
1	Maintainable	18	1	40.9	13.6	16.76
2	Maintainable	17	1	15.1	16.9	19.23
3	Maintainable	16	1	47.9	12.7	16.69
4	Maintainable	13	1	55.5	9.5	13.32
5	Maintainable	13	1	56.8	9.5	13.3
6	Maintainable	28	1	61	9.2	12.24
7	Maintainable	21	1	57.3	10.9	13.8
8	Maintainable	18	1	66	9.3	11.75
9	Maintainable	29	1	52.9	10.4	13.98
10	Maintainable	15	1	41.7	12.7	16.68
11	Maintainable	16	1	18.3	15.9	19.79
12	Unmaintainable	21	18	53.6	11.6	15.45
13	Unmaintainable	17	14	41.4	12.9	16.45
14	Unmaintainable	22	7	32.4	14.8	17.52
15	Unmaintainable	18	14	32.3	14.2	17.64
16	Unmaintainable	20	13	35.8	12.7	16.49
17	Unmaintainable	24	6	17.5	16.4	19.39
18	Unmaintainable	14	13	63.3	9.5	13.03
19	Unmaintainable	17	13	42.8	13.7	16.62
20	Unmaintainable	17	13	59.5	10.9	13.42
21	Unmaintainable	26	18	25.2	15.3	20
22	Unmaintainable	18	11	39.7	14.3	17.44
23	Unmaintainable	28	20	23.7	15.2	19.39
24	Unmaintainable	22	18	53.1	9.5	12.74
25	Unmaintainable	26	18	23.1	15.3	19.22
26	Unmaintainable	27	21	18.7	16	19.35
27	Unmaintainable	31	21	35.6	15	17.7
28	Unmaintainable	17	9	24.4	14.9	18.4
29	Unmaintainable	23	7	38.6	12.9	16.5