

In this issue:

- 4. Writing Prompts to Identify At-Risk Students in Introductory Programming Courses**
Jon D. Clark, Colorado State University
Seth J. Kinnett, Colorado State University

- 16. When One Account Exposes Millions: Design Debt, Relational Exposure, and the 23andMe Breach**
David J. Yates, Bentley University
Arthur Ream III, Bentley University

- 32. Implementing Personalized Learning Pathways as Informed by the 5E Model: Digital Tools to the Rescue!**
Celeste Tipple, LaTrobe University
Tanya Linden, The University of Melbourne

- 48. How 21st Century Skills Have Evolved in the 21st Century and How AI Is Shaping Their Next Evolution**
Mark Frydenberg, Bentley University
Kevin Mentzer, Nichols College

- 62. Teaching Case**
Paddles for Paws: Development of a Pickleball Tournament Event Management Database for a Cool Cause
Dana Schwieger, Southeast Missouri State University

- 75. Teaching Case**
From Concept to Canvas: Leveraging Generative AI to Co-Design Business Visuals
Fang Chen, University of Montana
Bryan Hammer, University of Montana
Shawn Clouse, University of Montana
Patricia Akello, University of Montana

The **Information Systems Education Journal** (ISEDJ) is a double-blind peer-reviewed academic journal published by **ISCAP** (Information Systems and Computing Academic Professionals). Publishing frequency is five times per year. The first year of publication was 2003.

ISEDJ is published online (<https://isedj.org>). Our sister publication, the Proceedings of the ISCAP Conference (<https://iscap.us/proceedings>) features all papers, abstracts, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the ISCAP conference. All papers, whether award-winners or not, are invited to resubmit for journal consideration after applying feedback from the Conference presentation. Award winning papers are assured of a publication slot; however, all re-submitted papers including award winners are subjected to a second round of three blind peer reviews to improve quality and make final accept/reject decisions. Those papers that are deemed of sufficient quality are accepted for publication in the ISEDJ journal. Currently the target acceptance rate for the journal is under 35%.

Information Systems Education Journal is pleased to be listed in the Cabell's Directory of Publishing Opportunities in Educational Technology and Library Science, in both the electronic and printed editions. Questions should be addressed to the editor at editor@isedj.org or the publisher at publisher@isedj.org. Special thanks to volunteer members of ISCAP who perform the editorial and review processes for ISEDJ.

2026 ISCAP Board of Directors

Amy Connolly
James Madison University
President

Michael Smith
Georgia Institute of Technology
Vice President

Jeff Cummings
Univ of NC Wilmington
Past President

David Firth
University of Montana
Director

Mark Frydenberg
Bentley University
Director/Secretary

Leigh Mutchler
James Madison University
Director

RJ Podeschi
Millikin University
Director/Treasurer

Bryan Reinicke
Rochester Institute of
Technology / Director

Jeffry Babb
West Texas A&M University
Director/Curricular Matters

Eric Breimer
Siena University
Director/2026 Conf Chair

Thomas Janicki
Univ of NC Wilmington
Director/Meeting Planner

Xihui "Paul" Zhang
University of North Alabama
Director/JISE Editor

Copyright © 2026 by Information Systems and Computing Academic Professionals (ISCAP). Permission to make digital or hard copies of all or part of this journal for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial use. All copies must bear this notice and full citation. Permission from the Editor is required to post to servers, redistribute to lists, or utilize in a for-profit or commercial use. Permission requests should be sent to Kevin Mentzer, Editor, editor@isedj.org.

INFORMATION SYSTEMS EDUCATION JOURNAL

Editors

Kevin Mentzer
Editor
Nichols College

Ira Goldman
Associate Editor
Siena University

David Yates
Associate Editor
Bentley University

Michelle Louch
Teaching Cases & Exercises
Editor
University of Pittsburgh - Greensburg

Mark Pisano
Teaching Cases & Exercises
Associate Editor
Southern Connecticut
State University

Thomas Janicki
Publisher
Univ of NC Wilmington

David Woods
Assistant Publisher
Miami University
Regionals

Paul Witman
Emeritus Editor
(2021-2026)
California Lutheran
University

Jeffry Babb
Emeritus Editor
(2016-2021)
West Texas A&M
University

Donald Colton
Emeritus Editor
(2003-2010)
Brigham Young University
Hawaii

Writing Prompts to Identify At-Risk Students in Introductory Programming Courses

Jon D. Clark
jon.clark@colostate.edu

Seth J. Kinnett
seth.kinnett@colostate.edu

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

Abstract

The identification of at-risk students early in introductory programming courses is critical to their success. Timely intervention requires assessment before substantial code has been written, and good and bad habits are formed. This study asserts that the use of natural language writing prompts can be used as a diagnostic tool, based on the SOLO taxonomy of cognitive development. The relationship between natural language metrics (Flesch Reading Ease, Flesch-Kincaid Grade Level, Gunning FOG Index) and code maintainability (McCabe's Essential Complexity) by 29 novice student programmers completing a Java assignment was evaluated. Statistical results show significant differences in all three natural language metrics between students who produced maintainable and unmaintainable code, with the strongest predictability demonstrated by the FOG Index (FOG Index: $p=.086$, $CI_{90} = [-3.4, -.20]$). These results were interpreted through the SOLO taxonomy, suggesting that students performing at the Multistructural level produce both disconnected writing (high complexity scores) and unstructured code (high essential complexity), while students performing at the Relational level produce coherent structures in both domains. Further, this study provides implementation guidelines for instructors to manage writing prompts, interpret results, and design impactful interventions, resulting in a low-cost, scalable approach for early student assessment.

Keywords: SOLO Taxonomy, Natural Language Metrics, Software Metrics, Halstead, McCabe, Programming Education

Recommended Citation: Clark, J.D., Kinnett, S.J., (2026). Writing Prompts to Identify At-Risk Students in Introductory Programming Courses. *Information Systems Education Journal*, v24(n3) pp 4-15. DOI# <https://doi.org/10.62273/RRRD8805>

Writing Prompts to Identify At-Risk Students in Introductory Programming Courses

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 (1988) 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.

Three useful and easily obtainable Natural Language (NL) metrics were chosen for this research. 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:

$$\text{Flesch Reading Ease} = 206.835 - 1.015(\text{total words/total sentences}) - 84.6 (\text{total syllables/total words})$$

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:

$$\text{Flesch-Kincaid Grade Level} = 0.39 (\text{Total Words/Total Sentences}) + 11.8 (\text{Total Syllables/Total Words}) - 15.59$$

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

$$\text{Fog Index} = 0.4 (\text{words/sentences}) + 100 (\text{complex words/words})$$

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 appropriate 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.

A graph of control flow 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) = $1 \leq$
 $ev(G) \leq v(G)$: based on reduced control
flow graph

Interpretation of $v(G)$ thresholds by McCabe:

1. 1-10: simple procedure, little risk
2. 11-20: more complex, moderate risk
3. 21-50: complex, high risk
4. >50: untestable code, very high risk

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. SOLO TAXONOMY OF COGNITIVE DEVELOPMENT

The Structure of Observed Learning Outcomes (SOLO) taxonomy (Biggs and Collis, 1982) identifies five hierarchical levels of cognitive complexity evidenced in learning: Structural, Unistructural, Multistructural, Relational, and Extended Abstract. Learning complexity, and arguably effectiveness increases in the hierarchy. This framework has demonstrated usefulness relative to assessing learning outcomes (Burnett, P.C., 1999) and in the assessment of student code comprehension, algorithm design, and problem solving (Lister, et.al., 2006). Additionally, this taxonomy has been evaluated in empirical studies of Boulton-Lewis, G.M. (1995) and Chan et.al. (2002). It's noteworthy that this taxonomy aligns well with the extension of Bloom's taxonomy (Krathwohl, 2002) with a more insightful approach to how learners integrate knowledge.

The five levels of SOLO identify increasing levels of sophistication and ability. The Prestructural level is one in which one misses the point entirely and may not know where to begin. The result is that to proceed one must choose a focus whether or not it turns out to be useful in the end. The second level is called Unistructural due to the fact that a focal point is chosen in order to reduce the level of chaos. Development beyond a single focus is a critical transition point relative to learning in which multiple focal points and their relationships are identified. This level predictably

is referred to as Multistructural. With greater mastery of the number of factors involved and the influence of relationships the Relational level is achieved. With this level of understanding much of the original chaos has been replaced with predictability. The final level is achieved when students can generalize and reorganize their understanding in other contexts. This level is called Extended Abstract.

We suggest that natural language complexity and code complexity are indicators of a learner's SOLO level.

Let's consider three students at different SOLO levels of complexity and the impact of their approach to an assignment. The Prestructural student who is facing an assignment involving Java, has to identify which statements are appropriate for a possible solution, and in addition, may not understand the problem. Maximum chaos exists and the only way to proceed is to focus on something. One might choose to understand the problem rather than the solution.

The Multistructural student on the other hand likely has developed an understanding of the problem, likely in a context without the programming language and all of its complexity. With this domain knowledge the student can greatly simplify the choices that have to be made to construct a solution and the language constructs that are useful for this purpose.

The Relational student will demonstrate a predictable approach that occurs in natural language expression, and the development of code. Likely, we will find that the learner will follow these steps:

1. Decompose a complex goal into manageable subgoals (outline in natural language equivalent to functions in an algorithm).
2. Organize elements into a logical sequence (paragraph flow equivalent to program control flow).
3. Establish clear relationship between parts (transitions of ideas versus function call sequences).

3. 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 <i>for</i> or <i>while</i> 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. Appendix I 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 and an external application for the Gunning FOG Index. The dataset consists of McCabe's Cyclomatic 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 I has been partitioned into the McCabe categories of Maintainable and Unmaintainable. These are highly significant and based on a threshold of 4 for Essential Complexity where if $ev(G) > 4$ the code is Unmaintainable for $ev(G) \leq 4$ 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 $v(G) \leq 10$ is Reliable 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.

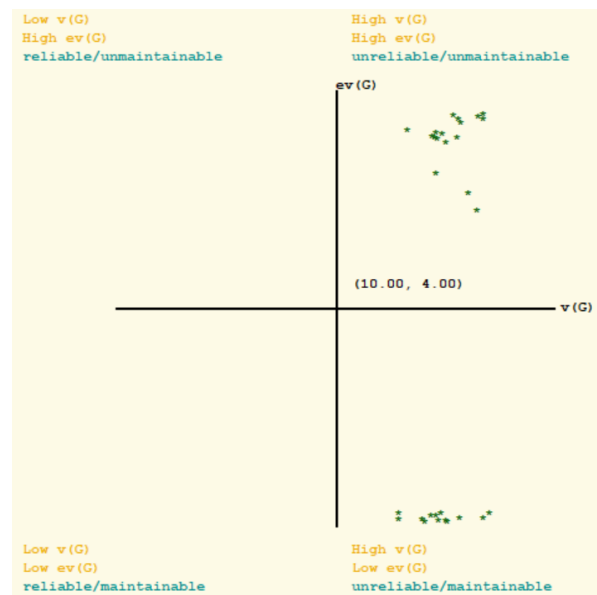


Figure 2: $v(G)$ & $ev(G)$ Scatterplot of Student Submissions

The distinction between Reliability and Maintainability may appear to be confusing. According to McCabe, the two subcategories of Reliability and Unreliability are 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 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 and explanation of these constructs.

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: What is the relationship between Flesch Reading Ease mean scores and UM and M categories determined by McCabe's ev(G)?

(H₀1) There is no difference in mean Flesch Reading Ease scores between the UM and M categories determined by McCabe's ev(G)

(H₁1) There is a difference in mean Flesch Reading Ease between the UM and M categories.

RQ2: What is the relationship between Flesch-Kincaid Grade Level mean scores across the UM and M categories determined by McCabe's ev(G)?

(H₀2) There is no difference in mean Flesch-Kincaid Grade Level scores across the UM and M categories determined by McCabe's ev(G)

(H₁2) There is a difference in mean Flesch-Kincaid Grade Level scores between the UM and M categories determined by McCabe's ev(G)

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

(H₀3) There is no difference in mean Gunning Fog Index scores between the UM and M categories determined by McCabe's ev(G)

(H₁3) 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.

One of the paper's authors administered the writing sample, while the other author administered the programming assignment. Both data collection exercises occurred through the natural administration of respective courses. Students included had completed both courses within one year of each other. The sample of programming exercises was drawn from a prior study, which sought to identify patterns in the McCabe metrics in introductory programming courses.

This population was identified first, then the writing samples collected as part of a different exercise were matched by name and the relevant scores were computed. The McCabe metrics used to segment programming samples into Maintainable and Unmaintainable categories were generated using the BattleMap IQ software. The readability metrics were generated using a combination of tools. The Flesch index and Grade Level index were computed using native tools in Microsoft Word (Review->Editor->Insights->Document Stats) while the FOG index was obtained via the public website <http://gunning-fog-index.com>.

Once these scores were all obtained, we created a spreadsheet containing two group indicators (Maintainable/Unmaintainable) and included columns for each of the three readability metrics. This data was loaded into SPSS Statistics v. 30.

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 ($N_{\text{maintainable}} = 11$, $N_{\text{unmaintainable}} = 18$). 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. In order to assess the magnitude of any practical effect commensurate with statistical differences, we also generated Hedges' g, which applies a correction factor to the pooled standard deviation to reduce bias in the estimation of population

effect sizes (Hedges, 1981). This measure was selected for its widespread use in meta-analytic research, facilitating cross-study comparability.

4. RESULTS

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 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. Table 3 shows the summary findings from our bootstrapped independent samples t-test. Hedge’s *g* indicates a medium to large practical effect size when comparing groups across all three metrics, with the highest being in Flesch-Kincaid Grade Level and FOG Index.

Metric	MD	σ_M	CI ₉₀	<i>g</i>
Flesh Index	9.97	5.95	[.04, 19.5]	.65
Grade Level	-1.7	.97	[-3.2, -.10]	-.72
FOG Index	-1.8	.98	[-3.4, -.20]	-.72

Table 3: Bootstrap Independent Samples Test

RQ1 asks *What is the relationship between the Flesch Reading Ease and McCabe metrics determining the UM and M categories?* The 90% confidence intervals do not span zero (CI₉₀ = [.039, 19.52]), leading us to reject H₀₁. 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 *What is the relationship between Flesch-Kincaid Grade Level and McCabe metrics determining the UM and M categories?* In this case, the CI₉₀ = [-3.26, -.10]. Once again, these confidence intervals do not span zero, leading us to reject H₀₂. 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 *What is the relationship between the Gunning Fog index and McCabe metrics determining the UM and M categories?* This case has CI₉₀ = [-3.4, -.20]. Similarly to the prior metrics, we are empowered by the findings to reject H₃₀ and conclude there is a significant difference in the FOG index mean scores across the groups. Further, we observe that among the three readability metrics, the FOG Index demonstrated the most robust group difference, with its 90% bootstrap confidence interval [-3.4, -0.20] excluding zero by the widest margin. This was further supported by a medium-to-large effect size (*g* = -.72). Again, the FOG 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 statistically significant differences across all three natural language metrics when comparing the two groups of maintainable vs. unmaintainable code based on McCabe’s *ev(G)*. Even though the practical effect sizes are notable based on Hedges’ *g*, the statistical findings retain validity at 90% confidence but not 95%. A significance level of $\alpha = .10$ (i.e., 90% confidence intervals) was adopted consistent with the exploratory nature of the study (Hair et al., 2009). The FOG index mean scores had the highest differences across the two groups which, along with a medium to high practical effect size via Hedges’ *g* reveals FOG score to be our preferred mechanism to recommend to instructors interested in performing these assessments.

5. 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.

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 are likely 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.

These findings are consistent with the known understanding of the SOLO taxonomy. Namely, students with high FOG scores, which could be seen as representing essentially a Multistructural level of understanding, generate natural language that is cumbersome to consume, despite generally following the norms of English language. Namely, a solution can be generated, but natural language written at this level is analogous to the *stream of consciousness* approach, where syntactic succinctness or cognition surrounding sentence structure is largely absent. Our findings demonstrate the relationship between this reality and high $ev(G)$ scores on an introductory programming assignment. Since $ev(G)$ represents the maintainability of a program, characterized by a control flow graph that excludes structured programming primitives – techniques one would expect to see extensively in an introductory programming assignment artifact – it is evident that higher levels of $ev(G)$ represent programs that use convoluted techniques rather than focused deployment of collections of fundamental programming techniques (n.b., with programming primitives entailing sequences, selection statements, and iteration constructs). The resultant code is thus aptly characterized as Multistructural and absent the type of cohesion and reflection necessary to create code that could be justly assessed at the Relational level. Pedagogically, all students received the same instruction, yet some students wrote maintainable code “naturally” while others didn’t.

The key tactical pedagogical implication of this research is the potential early identification of students more likely to operate the Multistructural level in a programming course compared to the Relational level. We suggest instructors implement the following practices. First, at the beginning of the term, students are asked to write a 250-500 word response to a brief prompt. In our study, the prompt involved three answers in English text (see Appendix II, CIS360 Reflection Paper 1: IT Professionals). It is critical that

students write this without the use of generative AI tools, so embedding the response inside of a tool like Lockdown Browser may be advisable. Next, instructors generate FOG index scores using <http://gunning-fog-index.com/>.

Based on the data we evaluated, we suggest that FOG scores above 15 correspond with Multistructural English language performance and accordingly suggest that this collection of students are candidates for early intervention. The score of 15 is provisional and we recommend instructors look for division in scores in their own samples should they choose to pursue this path. We also note that this is only one possible dimension through which an instructor could define an at-risk student. Accordingly, we suggest instructors would benefit from evaluating a variety of mechanisms to identify students, who could benefit from some intervention.

Since we did not specifically test different intervention strategies, we defer to instructors for their preferred courses of action, though some options include reinforcing available resource options such as tutors, help centers, and selected online resources.

We further note that student language ability may even be critical to successful use of generative AI tools for code assistance. 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 and 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.

6. CONCLUSIONS

Computer programming entails a mastery of language. In this paper, we have demonstrated a relationship between characteristics of an English language writing assignment with characteristics of Java program code. This relationship enables instructors to forecast aspects of student Java programming fluency – namely, performance at the Multistructural vs. Relational levels of the SOLO taxonomy – vis a vis English language characteristics of student writing samples.

The recent advent of vibe coding also suggests that English language mastery has both indirect benefits to direct program code but also via vibe prompts, where better writers are presumably generating better responses from LLMs.

In addition, further research may be needed to determine the pedagogical impact of coding practice. The UM category contains a great deal of unstructured code that deserves further study including:

- 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

- Boulton-Lewis, G. M. (1995). The SOLO taxonomy as a means of shaping and assessing higher-order learning in higher education. *Higher Education Research & Development*, 14(2), 143–154. <https://doi.org/10.1080/0729436950140201>
- Biggs, J. B., & Collis, K. F. (1982). *Evaluating the quality of learning: The SOLO taxonomy (structure of the observed learning outcome)*. Academic Press. <https://doi.org/10.1016/C2009-0-19712-5>
- Burnett, P. C. (1999). Assessing the structure of learning outcomes. *Educational Psychology*, 19(3), 309–318. <https://doi.org/10.1080/0144341990190305>
- Chan, C. C., Tsui, M. S., Chan, M. Y. C., & Hong, J. H. (2002). Applying the structure of the observed learning outcomes (SOLO) taxonomy on students' learning outcomes: An empirical study. *Assessment & Evaluation in Higher Education*, 27(6), 511–527. <https://doi.org/10.1080/0260293022000020282>
- Flesch, R. F. (1955). *Why Johnny can't read—and what you can do about it*. Harper & Brothers.
- Gunning, R. (1969). The fog index after twenty years. *Journal of Business Communication*, 6(2), 3–13. <https://doi.org/10.1177/002194366900600202>
- Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E. (2009). *Multivariate data analysis* (7th ed.). Prentice Hall.
- Halstead, M. H. (1977). *Elements of software science*. Elsevier.
- Hedges, L. V. (1981). Distribution theory for Glass's estimator of effect size and related estimators. *Journal of Educational Statistics*, 6(2), 107–128.
- Karpathy, A. (2025, February 2). Tweet. X (formerly Twitter). Retrieved May 9, 2025, from <https://twitter.com>
- Kincaid, J. P., Braby, R., & Mears, J. (1988). Electronic authoring and delivery of technical information. *Journal of Instructional Development*, 11(2), 8–13. <https://doi.org/10.1007/BF02904998>
- Kinnett, S. J., & Clark, J. D. (2024, November 6–9). A study of software metrics, student learning, and system development metrics. In *Proceedings of the ISCAP Conference* (Baltimore, MD).
- Krathwohl, D. R. (2002). A revision of Bloom's taxonomy: An overview. *Theory Into Practice*, 41(4), 212–218. https://doi.org/10.1207/s15430421tip4104_2
- Lister, R., Simon, B., Thompson, E., Whalley, J. L., & Prasad, C. (2006). Not seeing the forest for the trees: Novice programmers and the SOLO taxonomy. In *Proceedings of the 11th Annual Conference on Innovation and Technology in Computer Science Education (ITICSE '06)* (pp. 118–122). ACM. <https://doi.org/10.1145/1140124.1140157>

- McCabe Software. (2024, May 15). *The software path analysis company*. Retrieved March 18, 2026, from <https://mccabe.com>
- McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4), 308–320. <https://doi.org/10.1109/TSE.1976.233837>
- Ng, A. (2025, May 28). Replit launches vibe coding 101 for AI-driven application development. *Blockchain.News*. Retrieved March 18, 2026, from <https://blockchain.news/flashnews/replitlaunches-vibe-coding-101-for-ai-driven-application-development>
- Rescher, N. (1988). *Complexity: A philosophical overview*. Transaction Publishers.
- Riabov, V. V. (2007). Graph theory applications in developing software test strategies for networking systems. *River Academic Journal*, 3(1), 1–10.
- Sinnemäki, K. (2011). *Language universals and linguistic complexity: Three case studies in core argument marking* (Doctoral dissertation, University of Helsinki). Retrieved April 28, 2016, from <http://urn.fi/URN:ISBN:978-952-10-7259-8>

Appendix I. 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

Appendix II. CIS360 Reflection Paper 1: IT Professionals

Name: _____

Requirements: Read articles on "Role of the Systems Analyst" and watch the IT & Strategy video posted on the Module 1 Canvas page and answer the questions below. Submit a word-processed document on Canvas by the deadline.

Evaluation Criteria: Overall quality of your writing (including using your words rather than quotes from the articles); the correctness, thoughtfulness, and clarity of your responses.

1. IT-business alignment has been one of the top three concerns of IT managers for the last ten years (or more), according to a series of studies on the "IT Issues and Trends." Watch the IT & Strategy video posted on the Module 1 Canvas page.
 - a) Describe an example (from the readings, your experience, or the news) of an organization with strong IT-business alignment and an example of an organization with weak IT-business alignment and explain your rationale.ⁱ (1 paragraph)
 - b) IT-business misalignment is often attributed to the "troubled relationship" between IT and the rest of the organization. What is the cause of this troubled relationship, according to the video? Reflect on your own experiences, either as part of "IT" or as part of "the rest of the business" (e.g., as an employee in your current or previous job, as a customer who has had to deal with IT professionals for tech support). Do you agree or disagree with the authors, and why? Why do you think the IT-business relationship is often problematic? (1 paragraph)
 2. A systems analyst is an IT professional, and, as such, is often perceived as a "geek." However, I would argue that if the rest of the organization perceives the systems analyst as a one-dimensional technology expert, then the analyst should consider a different career (or should develop additional skills). After reading the description of systems analysts from the Bureau of Labor Statistics' Occupational Outlook Handbook and the IS Job Index 2019, how would you describe a systems analyst? What skills do you think an analyst needs, and why? (1 paragraph)
 3. Think about your career goals (short- and long-term), and the knowledge/skill sets you will need to achieve those goals. What do you want to do (short-term, long-term, or both), and what kind of organization do you want to work for? Will you be on the IT side or the business side (or both)? What knowledge/skill sets will you need to be successful? What are your current strengths and weaknesses? How well prepared do you think you will be when you graduate, and if you do not think you are getting the preparation you need, what else can/will you do? Finally, are you looking forward to your post-graduation career? What about it excites or concerns you?
-