Impact of Programming Language Features on Code Review Success

CS 485 - Introduction to Empirical Research in Software Engineering
(Group 4) James Gaiser and Dylan Dunsheath

Abstract: *All good software projects require extensive review to ensure that they are to the highest quality possible. But not all code reviews are the same and have varying levels of quality. In the effort to make reviews more successful, this research looks at the potential impact that the features of a programming language may have on code review success.*

Github: https://github.com/JamesG9802/Language-Feature-Code-Review

I. Introduction

Code review, often referred to as peer review, is a fundamental part of the software development process, acting to ensure the quality of the code base [1]. Typically, a reviewer will examine another person's code to statically analyze it for logical errors or nonoptimal structure to align with best practices. It is apparent that programmers would want to write better code to pass these code reviews as often as possible, or to have a higher code review "success".

As programming languages are a necessary part of all software development, languages are likely to be a key factor in affecting code review success. Our objective is to examine the impacts of a language's features and how reviewers' decisions may be altered. Understanding the relationship between language features and code review will be a useful factor in deciding what languages should be used in software projects. We will go through a review of preexisting literature, a detailing of our theoretical model, an overview of our methodology, and a summary of our results.

II. Literature Review

Programming Language Features

Frustratingly, programming language features, hereafter referred to as features, are discussed often in literature, yet are rarely actually defined in depth; the Oberon language debuts its type extension feature [2], the TraitRecordJ language details its hierarchy-free trait feature [3], and Rust language defines their safety-inspired ownership features [4]. Features are often used as an implied term that is never elaborated on. However, this lack of specificity is insufficient for our research. Is type-checking considered a feature? What about purely cosmetic developer-aids like comments? Before proceeding, it is necessary to establish an acceptable definition of programming language features.

It is apparent that features are inherently tied to programming languages. It is known that programming languages can be viewed through three different aspects [5, pp. 4-6]. These are:

1. Syntax—the form of a language.

2. Semantics—the meaning of the language.
3. Pragmatics—how the language is implemented.

It is also known that programming languages can be decomposed into three distinct parts [5, pp. 207-208]. These are:

1. Kernel—the functional core of a language.
2. Syntactic sugar—syntactic constructs that make it easier to access the kernel.
3. Standard library—the procedures, constants, and operators of a language.

With this in mind, we suggest an inclusive definition for programming language features as follows—a programming language feature is a capability of a programming language provided through its syntax, semantics, or pragmatics that extends the language's kernel, syntactic sweetness, or standard library.

Code Review

An ACM paper [6] found that code reviews are influenced by a variety of factors, most important of which is code quality. Code quality itself can be defined in terms of readability (how easy it is to read and understand what the code is doing), simplicity (how easy it is to follow along with the code), and maintainability (how easy it is to update and change the code). It also makes sense for code quality to be associated with the correctness of the code, or how well it actually works and functions like it is supposed to.

Besides the code itself, reviews are also impacted by interpersonal factors. The same ACM paper noted that "If it's someone you trust you don't have to check things as rigorously" [6]. The time reviewers take, the experience of both the reviewer and reviewee, and the relationship between the reviewers are all influential factors. Finally, it is important to acknowledge that code reviews themselves are not perfect. Up to 54% of reviewed changes introduced bugs into the codebase, indicating that they failed to properly evaluate the code.

In building our theoretical framework and study design, we must keep all these factors in mind to ensure our results are accurate.

III. Theoretical Framework

Dependent Variable

The dependent variable we are examining in our study is code review success, which we have defined as a review where the reviewer approves of the code additions or changes by a contributor.

Independent Variable

The independent variable we are studying is programming language features, which we previously defined to be viewed through three lenses—syntax, semantics, and pragmatics—along with its distinct parts—kernel, syntactic sugar, and standard library. In our study, we will mainly be examining the kernel and syntactic sugar only, as in several languages the shared library tends to have the same abilities such as I/O and mathematical functions [7].

Moderating Variables

Like with existing literature on the topic, the moderating variables in our study are qualitative variables represented by the demographics of the respondents. We have chosen to include their experience programming, reviewing code, and having their own code reviewed.

Mediating Variable

The mediating variable in our study is code quality. Following the ACM study's results [6], code quality can be broken into different aspects: readability, maintainability, and simplicity. Considering the fact that code does not always work as intended, we additionally add correctness as another indicator of code quality.

Hypothesis

Our hypothesis is that programming language features are statistically significant in affecting the success of a code review. This was justified from two case studies we examined: JavaScript vs. TypeScript and C vs. Rust.

The JavaScript language is the most used and desired language in 2023 [8]. A defining feature of the language is its use of dynamic and weak types for convenient and concise code. However, these can lead to unintuitive implicit coercions that "create subtle bugs when conversions happen when they are not expected, or where they are expected in the other direction" [9]. TypeScript is a variant of JavaScript that gives the developer the option to remove dynamic and weak typing [10]. It is technically a more restrictive version of JavaScript; yet compared to JavaScript's 57.83% admiration rate—a measure of how many people want to continue using the language—TypeScript has a much more impressive 71.70% admiration rate. This is an example of how a language feature can negatively impact code quality.

The C language is a relatively popular language among developers learning to code but not with professional developers. 32% of learners use C compared to 17% of developers [8]. This could be due to the fact that C has various language features that increase its complexity: its lack of classes, unsafe memory management system, and lack of exception handling [11]. On the other hand Rust does have classes, an ownership memory management system, and error handling [12]; it is the most admired language of 2023 with 84.66%.

We find these case studies justify our hypothesis and our rationale that programming language features affect code review success.

IV. Methodology

Survey Design

To fully capture our research requirements, we built a custom website to collect survey responses. Respondents were all first asked for their experience with programming and reviewing code using a 6-point likert scale.

Then they were asked to review up to six random code snippets from a pool utilizing various language features on the basis of readability, simplicity, maintainability, and correctness also using a 6-point likert scale[1]. They are written in C and utilize radix sorting.

[1] In our GitHub we provided all the code snippets we used. In our survey, only the third variation of wrong code snippets, labelled code_3.c in their respective folders were used.

Respondents could submit their responses early if they did not want to review all six code snippets. They were also asked if they would approve the code in a code review. Finally respondents had the amount of time it took to review the code passively recorded.

All respondents' first code snippet would be a control code snippet that utilized only generic language features commonly found in many programming languages. We define generic language features as the following[13]:

- primitive literals
- variables of primitive or object class
- exceptions
- input/output
- control flow (if/else)
- loop control
- methods or functions
- classes or interfaces
- comments
- collections
- packages, namespaces, or file organization systems

Code snippets came in two variations: a correct and wrong version. All the wrong code snippets have the same error with the outer loop variable swapped with the inner loop variable, causing the program to malfunction. The subset of code snippets that respondents could review was configured to ensure a respondent could not see the correct and incorrect version of the same code snippet.

Survey Subjects

We sent our survey to college Discord servers, for both the general student body and those majoring programming-related disciplines. Most of the servers were advertised for mainly New Jersey Institute of Technology students. After around a month and half we analyzed our ($n = 26$) survey responses.

Software

Statistical analysis was performed through IBM SPSS Statistics. Microsoft Office Excel 365 was used for visualization.

Statistical Analysis

Our first step was to clean the data up for malicious and incomplete responses. We define malicious responses as those that include deliberately inaccurate data, such as those with negative years of experience or unrealistically years of experience (≥ 100). Additionally, we filter out responses that did not answer any question at all. This leaves us with ($n = 23$) valid responses.

We performed reliability analysis using Cronbach's Alpha. Correlations were examined with Pearson's correlation coefficient. As we are trying to determine the impacts of language features on code review success, we utilized one sample T-tests to determine if the scores given to code snippets utilizing language features had a statistically significant difference compared to the control.

This study presents the mean, minimum, maximum, and standard deviation of the data. Finally, we define statistical significance as a P-value below 0.05.

V. Results

Respondents' Demographic

On average, respondents tended to have more total years of programming experience and comparatively little professional (Table 1).

Python was the most popular language with almost 70% of the sample reporting having worked with it (Figure 2).

Respondents tended to be more experienced having their own code reviewed rather than reviewing other people's code (Table 1).

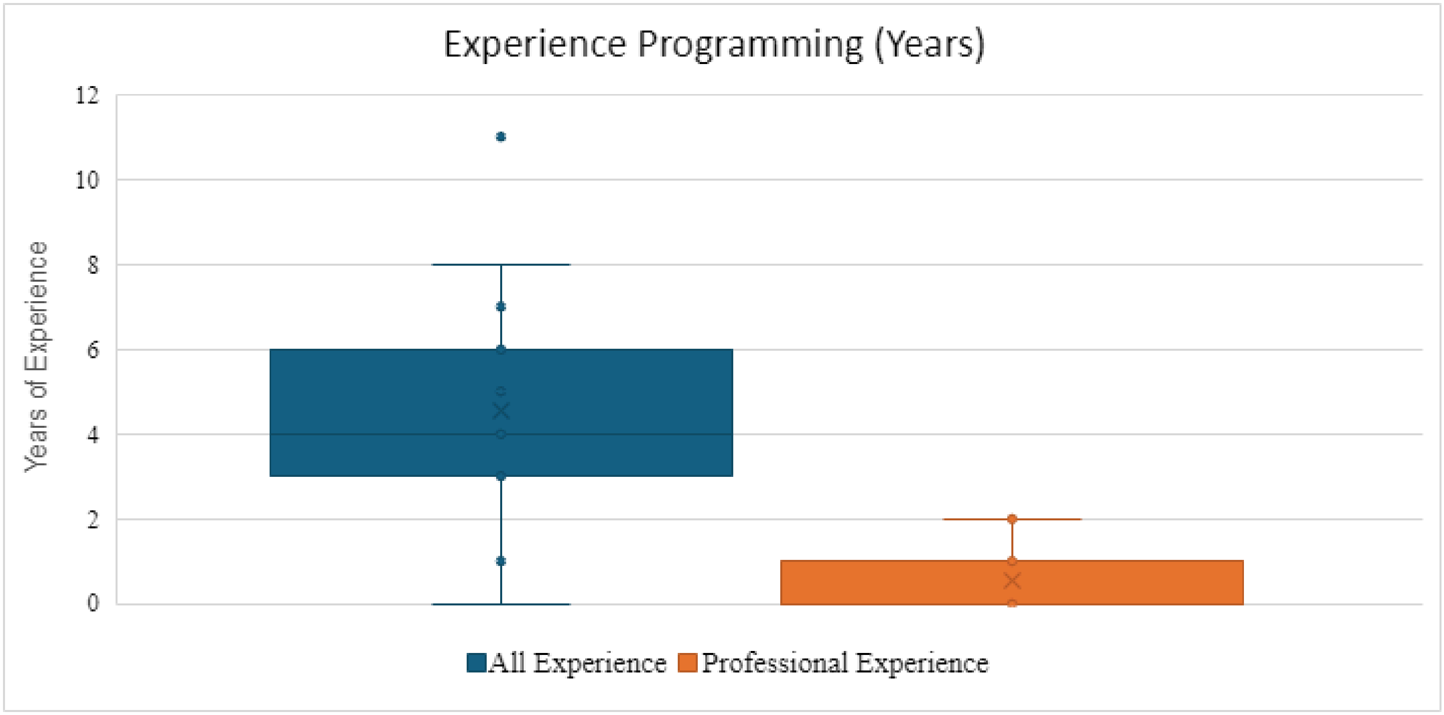


Fig. 1 A box and whiskers plot of the experience from respondents.

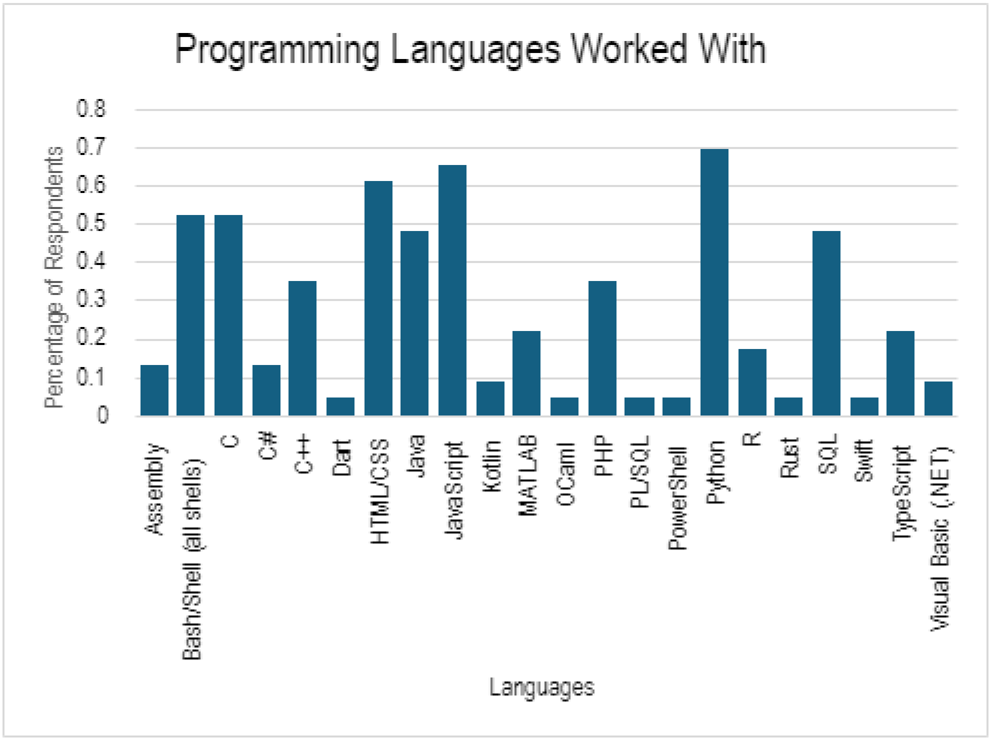


Fig. 2 A bar chart showing what percentage of the respondents worked with a language.

Table 1 A table showing the descriptive statistics of the respondents' demographic data.

Metric	Minimum	Maximum	Mean	Standard Deviation
All Experience	0	11	4.57	2.253
Professional Experience	0	2	0.57	0.788
Reviewer Experience	1	6	3.48	1.563
Reviewee Experience	1	6	4.09	1.505

Code Snippets

Table 2 Correct code snippets using general language features (the control group).

Metric	Minimum	Maximum	Mean	Standard Deviation
Readability	1	6	4.77	1.363
Simplicity	1	6	4.08	1.320
Maintainability	3	6	4.85	0.899
Correctness	3	6	4.38	0.961
Approval	1	6	4.23	1.363

Table 3 Wrong code snippets using general language features (the control group).

Metric	Minimum	Maximum	Mean	Standard Deviation
Readability	1	6	4.20	1.874
Simplicity	1	5	3.70	1.767
Maintainability	1	6	4.10	1.729
Correctness	1	6	3.90	1.663
Approval	1	6	3.50	1.780

Table 4 Correct code snippets using language features (the experimental group).

Metric	Minimum	Maximum	Mean	Standard Deviation
Readability	1	6	4.2308	1.35326

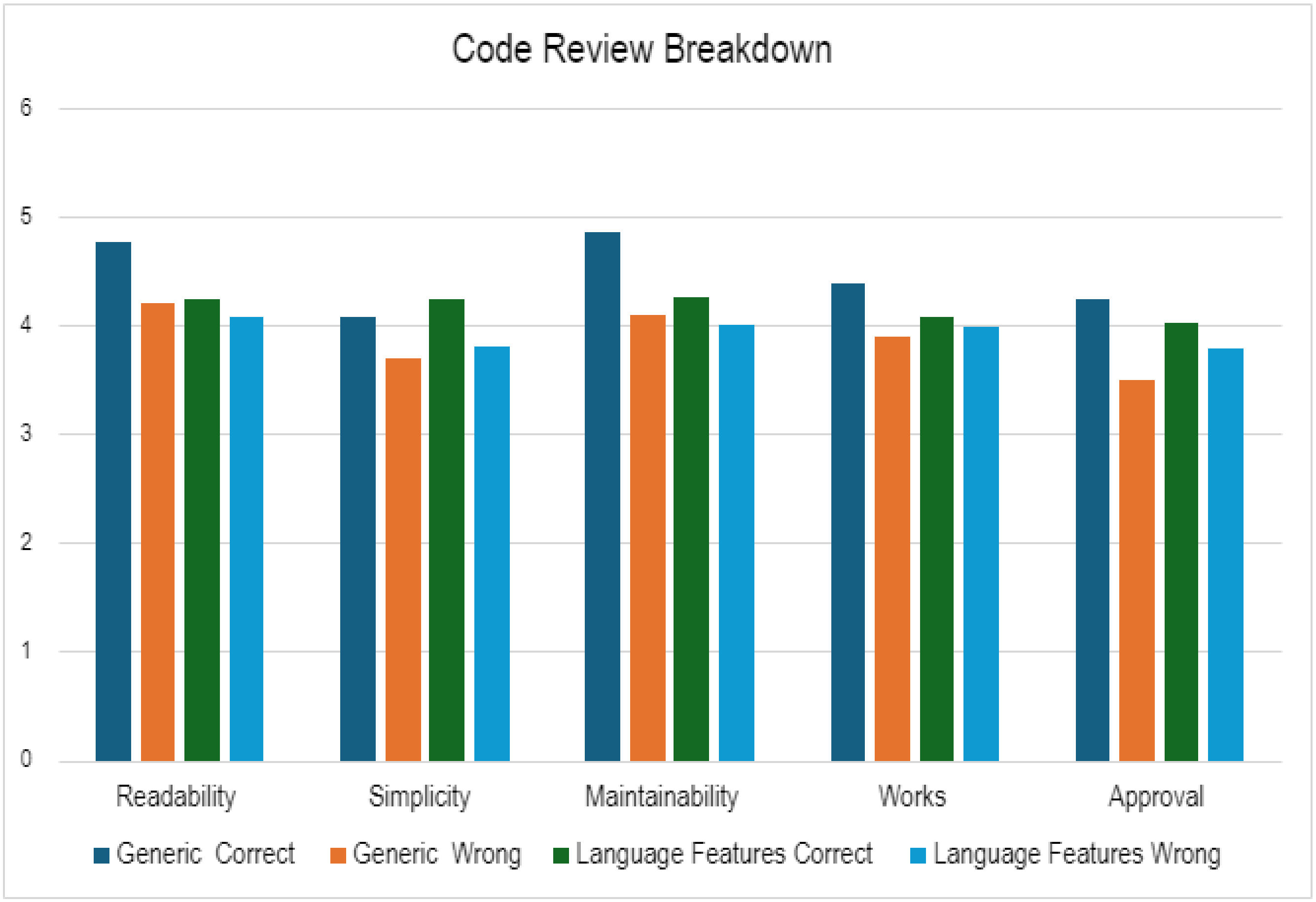
Simplicity	1	6	4.2308	1.15382
Maintainability	1	6	4.2564	1.27237
Correctness	1	6	4.0769	1.01872
Approval	1	6	4.0256	1.22894

Table 5 Wrong code snippets using language features (the experimental group).

Metric	**Minimum**	**Maximum**	**Mean**	**Standard Deviation**
Readability	1	6	4.0667	1.47235
Simplicity	1	6	3.8000	1.20793
Maintainability	1	6	4.0000	1.24316
Correctness	1	6	3.9778	0.98448
Approval	1	6	3.7778	1.06541

Respondents scaled each metric for the control code snippets on a six-point likert scale where 1 represented “Strongly Disagree” and 6 represented “Strongly Agree”. For both the correct and incorrect versions of both the control and experimental code snippets, the average score for every metric was $>= 3.5$, showing respondents agreed in varying strengths with the code snippet’s readability, simplicity, etc. On average, the correct control snippet scored the highest on all metrics, while the incorrect control code snippet scored the worst on every metric besides maintainability.

Fig. 3 Bar chart showing the scores each type of code snippet received on average for each metric.



Reliability

With a Cronbach's Alpha test, the reliability of the code snippet's metrics were evaluated. All four types of code snippets received a reliability score above > 0.85 (Table 6), indicating that the results can be trusted.

Table 6 Reliability of each code snippet's metrics.

Question Type	**Reliability**
Control (Correct)	0.889
Control (Wrong)	0.853
Language Feature (Correct)	0.926
Language Feature (Wrong)	0.903

Correlation

Table 7 Correlation between the correct control code snippet's metrics.

		Readability	Simplicity	Maintainability	Correctness	Approval
Readability	R	1	0.844	0.853	0.646	0.703

	Significance		<.001	<.001	0.017	0.007
Simplicity	R	0.844	1	0.713	0.632	0.406
	Significance	<.001		0.006	0.021	0.169
Maintainability	R	0.853	0.713	1	0.557	0.507
	Significance	<.001	0.006		0.048	0.077
Correctness	R	0.646	0.632	0.557	1	0.499
	Significance	0.017	0.021	0.048		0.082
Approval	R	0.703	0.406	0.507	0.499	1
	Significance	0.007	0.169	0.077	0.082	

Table 8 Correlation between the wrong control code snippet's metrics.

		Readability	Simplicity	Maintainability	Correctness	Approval
Readability	R	1	0.423	0.851	0.506	0.633
	Significance		0.223	0.002	0.135	0.049
Simplicity	R	0.423	1	0.629	0.178	0.124
	Significance	0.223		0.051	0.623	0.734
Maintainability	R	0.851	0.629	1	0.545	0.596
	Significance	0.002	0.051		0.103	0.069
Correctness	R	0.506	0.178	0.545	1	0.882
	Significance	0.135	0.623	0.103		<.001
Approval	R	0.633	0.124	0.596	0.882	1
	Significance	0.049	0.734	0.069	<.001	

Table 9 Correlation between the correct experimental code snippet's metrics.

		Readability	Simplicity	Maintainability	Correctness	Approval
Readability	R	1	0.794	0.775	0.477	0.574
	Significance		<.001	<.001	0.002	<.001
Simplicity	R	0.794	1	0.936	0.783	0.757

	Significance	<.001		<.001	<.001	<.001
Maintainability	R	0.775	0.936	1	0.641	0.648
	Significance	<.001	<.001		<.001	<.001
Correctness	R	0.477	0.783	0.641	1	0.867
	Significance	0.002	<.001	<.001		<.001
Approval	R	0.574	0.757	0.648	0.867	1
	Significance	<.001	<.001	<.001	<.001	

Table 10 Correlation between the wrong experimental code snippet’s metrics

		Readability	Simplicity	Maintainability	Correctness	Approval
Readability	R	1	0.791	0.836	0.304	0.427
	Significance		<.001	<.001	0.042	0.003
Simplicity	R	0.791	1	0.908	0.55	0.683
	Significance	<.001		<.001	<.001	<.001
Maintainability	R	0.836	0.908	1	0.582	0.669
	Significance	<.001	<.001		<.001	<.001
Correctness	R	0.304	0.55	0.582	1	0.887
	Significance	0.042	<.001	<.001		<.001
Approval	R	0.427	0.683	0.669	0.887	1
	Significance	0.003	<.001	<.001	<.001	

The correct control code snippet’s approval, or code review success, is significantly correlated ($p < 0.05$) with only readability (Table 7). Likewise, the wrong control code snippet’s approval is also correlated ($p < 0.05$) with readability as well as correctness (Table 8). For the experimental code snippets, both the correct and wrong code snippets were significantly correlated ($p < 0.05$) with readability, simplicity, maintainability, and correctness.

Statistical Significance

Table 11 Correlation between the wrong experimental code snippet’s metrics

Metric	**t**	**df**	**Two-Sided P**

Readability (Correct)	-2.488	38	0.017
Simplicity (Correct)	0.816	38	0.420
Maintainability (Correct)	-2.913	38	0.006
Correctness (Correct)	-1.858	38	0.071
Approval (Correct)	-1.038	38	0.306
Readability (Wrong)	-0.607	44	0.547
Simplicity (Wrong)	0.555	44	0.581
Maintainability (Wrong)	-0.540	44	0.592
Correctness (Wrong)	0.530	44	0.599
Approval (Wrong)	1.749	44	0.087

The experimental group showed statistically significant differences to the control in several areas. For correct code snippets, the experimental group showed a significant $(p < 0.05)$ decrease in readability and maintainability compared to the control (Table 11). No significant differences between the scores for the wrong experimental group compared to the control.

VI. Discussion

Our target audience roughly matches the expected values compared to previous studies. Like in the Stack OverFlow 2023 Developer survey, the top three used languages by people learning to code were HTML/CSS, JavaScript, and Python [8]. Considering our intended audience consisted of college students, it also makes sense that the majority of responses consisted of people who had vastly more total experience programming and comparatively little actual professional experience. Anecdotally, students spent more time writing code for assignments rather than looking over other people's code in group assignments, which helps explain why on average respondents were less familiar with reviewing other people's code compared to having their own code reviewed.

For the control code snippets, it makes logical sense that it would score higher than the wrong variation on all metrics, although it is surprising that the wrong variation did still score positively, although in our literature review it has been noted that code reviews were not perfect [6]. Both the correct and wrong variations' code review success were significantly correlated with readability, but only the wrong variation was correlated with correctness. This also makes sense because reviewers would not want to approve code with glaring mistakes or errors.

For the experimental code snippets, there was a correlation between code review success and every metric (readability, simplicity, maintainability, and correctness). Only the correct variation showed a statistically significant difference compared to the control; the readability and maintainability scores for correct experimental code snippets were lower than the control.

Based on the results, we should accept our alternative hypothesis that language features do have a significant impact on code review success. However, this relationship has only been strongly observed in code snippets that are correct.

VII. Conclusion

Our experiments indicate that there is a negative relationship between language features and code review success. Therefore, it may be beneficial for programmers to more strongly consider the language used for programming based on the features it possesses.

Future studies can expand on our work by examining the impacts of the various types of language features independently. Additionally it is known that code review is a more social interaction where the reviewer and reviewee have preexisting relationships and knowledge of each other. More research can be conducted to narrow down on the correlation between language features and code review success.

References

[1] GitLab, “What is a Code Review?,” *GitLab*, 2023. https://about.gitlab.com/topics/version-control/what-is-code-review/

[2] N. Wirth, “The Programming Language Oberon,” *Software: Practice and Experience*, vol. 18, no. 7, pp. 671–690, Jul. 1988, doi: https://doi.org/10.1002/spe.4380180707.

[3] L. Bettini, F. Damiani, I. Schaefer, and Fabio Strocco, “TraitRecordJ : A programming language with traits and records,” *Science of Computer Programming*, vol. 78, no. 5, pp. 521–541, May 2013, doi: https://doi.org/10.1016/j.scico.2011.06.007.

[4] W. Crichton, G. Gray, and S. Krishnamurthi, “A Grounded Conceptual Model for Ownership Types in Rust,” *Proceedings of the ACM on programming languages*, vol. 7, no. OOPSLA2, pp. 1224–1252, Oct. 2023, doi: https://doi.org/10.1145/3622841.

[5] Franklyn Albin Turbak, D. K. Gifford, and M. A. Sheldon, *Design concepts in programming languages*. New Delhi.: Phi Learning, 2010.

[6] O. Kononenko, O. Baysal, and M. W. Godfrey, ‘Code Review Quality: How Developers See It’, in *Proceedings of the 38th International Conference on Software Engineering*, Austin, Texas, 2016, pp. 1028–1038.

[7] H. R. Ahmad, M. Idrees, A. Ahmad, M. A. Butt, S. Shahzad, and M. Shahzad, ‘Unification of Programming Languages’, *Webology*, vol. 19, no. 3, pp. 423–457, 2022.

[8] Stack Overflow, “Stack Overflow Developer Survey 2023,” *Stack Overflow*, 2023. https://survey.stackoverflow.co/2023/

[9] MDN, “JavaScript data types and data structures,” *MDN Web Docs*, Oct. 23, 2019. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures

[10] Microsoft. “JavaScript with Syntax for Types.” *TypeScript*, www.typescriptlang.org/. Accessed 22 Apr. 2024.

[11] B. W. Kernighan and D. M. Ritchie, *The C programming language / ANSI C Version*, 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1988.

[12] S. Klabnik and C. Nichols, *The Rust Programming Language, 2nd Edition*. No Starch Press, 2022.

[13] H. R. Ahmad, M. Idrees, A. Ahmad, M. A. Butt, S. Shahzad, and M. Shahzad, ‘Unification of Programming Languages’, *Webology*, vol. 19, no. 3, pp. 423–457, 2022.