Article

Not peer-reviewed version

Does ChatGPT Help Novice Programmers Write Better Code? Results from Static Code Analysis

Philipp Haindl * and Gerald Weinberger

Posted Date: 17 June 2024

doi: 10.20944/preprints202406.1151.v1

Keywords: Programming Education; ChatGPT Large Language Models; Static Code Analysis

Article

Does ChatGPT Help Novice Programmers Write Better Code? Results from Static Code Analysis

Philipp Haindl * and Gerald Weinberger

Department of Computer Science and Security, St. Pölten University of Applied Sciences, Campus Platz 1, 3100 St. Pölten, Austria; philipp.haindl@fhstp.ac.at (P.H); gerald.weinberger@fhstp.ac.at (G.W.)
* Correspondence: philipp.haindl@fhstp.ac.at

Abstract: In the rapidly evolving landscape of AI-supported programming education, there's a growing interest in leveraging such tools to support students helping about what good code makes out. This study investigates the impact of ChatGPT on code quality among part-time undergraduate students enrolled in introductory Java programming courses. With no prior Java experience, students from two separate groups completed identical programming exercises emphasizing coding conventions and code quality. Utilizing static code analysis tools, we assessed adherence to a common coding convention ruleset and calculated cyclomatic and cognitive complexity metrics for submitted code. Our comparative analysis highlights significant improvements in code quality for the ChatGPT-assisted group (treatment group), with marked reductions in rule violations and both cyclomatic and cognitive complexities. Specifically, the treatment group demonstrated greater adherence to coding standards, with fewer violations across several rules, and produced code with lower complexity. These results suggest that ChatGPT can be a valuable tool in programming education, aiding students in writing cleaner, less complex code and better adhering to coding conventions. However, the study's limitations, including the small sample size and the novice status of the participants, necessitate further research with larger, more diverse populations and in different educational contexts.

Keywords: Programming Education; ChatGPT Large Language Models; Static Code Analysis

1. Introduction

ChatGPT, an advanced large language model (LLM) developed by OpenAI [1], has emerged as a significant tool capable of generating human-like text based on the input it receives. Since its introduction, ChatGPT has been leveraged across various domains, including coding and programming education. The capacity of ChatGPT to understand and generate code snippets in response to natural language prompts signifies a transformative potential for educational practices, especially in enhancing learning experiences and outcomes in programming courses. As the era of ChatGPT unfolds, programming education must identify the most effective ways to integrate and apply chatbots in the classroom. It is essential that students possess the ability to evaluate the quality of AI-generated code while also enhancing their programming fluency and critical thinking skills [2].

Code quality is a critical aspect of software development, encompassing properties like code structure, code layout, and statement quality. Previous studies have analyzed the static code quality of student code [3–5], common mistakes made by students when learning to program [6], and the semantic style of code [7].

This study examines the impact of ChatGPT on the quality of code produced by students in an introductory Java course. We employed static code analysis to detect rule violations to Java coding standards as well as cyclomatic and cognitive complexity in the source code of two groups and analyzed the deviations in the measurements. One group of students employed ChatGPT to implement the course's programming assignments, whereas the other group was devoid of access to ChatGPT. Static code analysis tools such as Checkstyle [8] and PMD [9] have also been used in previous studies to ensure code in a project conforms to a defined coding standard style and to identify flaws that could manifest themselves as bugs.

 doi:10.20944/preprints202406.1151.v1

The motivation behind this study is to provide insights into the implications of ChatGPT on code quality, particularly in an educational setting. Our findings may help educators tailor teaching materials to promote students' awareness of code quality and code understandability when using ChatGPT. To investigate the effect of ChatGPT onto code quality and code understandability among students in an introductory Java course, we formulated the following three research questions:

- **RQ1.** What violations to coding conventions [10] can be observed in the source code if students use or not use ChatGPT for the course's programming exercises?
- **RQ2.** What are the most violation-dense topics in the programming exercises if students use or not use ChatGPT for implementing them?
- **RQ3.** How does using ChatGPT influence the cyclomatic and cognitive complexity of the students' programming exercises' source code?

This paper is organized as follows: Section 2 delves into the related studies, while Section 3 outlines the methodology employed in our research. Afterwards, in Section 4, the results of our study are presented and specifically their implications for programming education discussed in Section 5. Consequently, in Section 6, the potential threats to the validity of our study are addressed, before concluding our work and suggesting avenues for future research in Section 7.

2. Related Work

We identified three streams of related research. The first stream involves investigating the quality of ChatGPT-generated code. In a study conducted by Li *et al.* [11], the authors examined the differences between human-authored code and code generated by ChatGPT. They developed a discriminative feature set and a dataset cleansing technique to differentiate between the two sources and achieved high accuracy in distinguishing ChatGPT-generated code from human-authored code. In another study by Guo *et al.* [12], the potential of ChatGPT in automating the code review process and improving the quality of student-generated code was evaluated. The results showed that ChatGPT outperforms *CodeReviewer* in code refinement tasks, but the authors also identified challenges and limitations, such as refining documentation and functionalities due to a lack of domain knowledge, unclear location, and unclear changes in the review comments. Liu *et al.* [13] systematically studied the quality of ChatGPT-generated code and found that although it can generate correct code for a significant number of tasks, there are still issues with wrong outputs, compilation or runtime errors, and maintainability. The study also revealed that ChatGPT's performance drops significantly on new programming tasks. The authors concluded that ChatGPT can partially address these challenges and improve code quality by more than 20%.

The second stream of research examines the code quality of novice programmers. Studies by Izu and Mirolo [4] and Börstler *et al.* [14] revealed disparities in the definition and prioritization of code quality among novice programmers, with aspects such as performance, structure, conciseness, and comprehensibility being evaluated differently. Izu and Mirolo [4] explored novice computer science students' perceptions of code quality, finding that students prioritize performance, structure, conciseness, and comprehensibility. The authors suggested that teaching should encourage discussion of best practices and personal preferences, rather than simply proposing a set of rules. Börstler *et al.* [14] investigated perceptions of code quality among students, educators, and professional developers. The authors observed that there was no common definition of code quality among the groups, and that readability was the most frequently named indicator of code quality.

Several studies have investigated the nature of common coding mistakes and quality issues among novice programmers. Several studies [15–18] provide insights into the frequent types of errors and code quality problems students encounter, including neglect of coding conventions and difficulties with program flow and modularization. These studies collectively underscore the importance of targeted instruction and feedback in helping students overcome these challenges and highlight the potential role of automated tools in educating students about code quality issues. Studies by Östlund *et al.* [5] and Brown and Altadmri [19] point to the impact of teaching assistants and changing instructional practices

 doi:10.20944/preprints202406.1151.v1

on improvements in code quality. However, Breuker *et al.* [3] revealed that despite these efforts, there is no significant difference in code quality between first-year and second-year students, suggesting a plateau in learning that needs to be addressed through more effective teaching strategies. Brown and Altadmri [19] conducted a study to determine the frequency of common mistakes made by students learning Java programming and to assess the accuracy of educators' estimations of these mistakes. By analyzing data from nearly 100 million compilations across over 10 million programming sessions from novice Java programmers, as well as a survey among educators, the researchers discovered a discrepancy between the estimates of educators and the actual mistakes made by students. This finding implies that educators may need to enhance their understanding of common students' mistakes to improve their teaching effectiveness.

The last stream of related research explores the potential of LLM-based chatbots such as ChatGPT in programming education. In a recent study, Kazemitabaar *et al.* [20] examined the suitability of *OpenAI Codex* on supporting introductory programming for novice programmers. The research aimed to determine whether novices could comprehend, modify, or extend code generated by these tools without developing a reliance on technology. The study, which involved young learners with no prior experience in text-based programming, found that LLM-based code generators significantly improved code-authoring performance, suggesting that such tools do not necessarily result in reliance on technology for coding tasks. Jacques [21] investigated the influence of ChatGPT on the quality of student-generated code in an introductory course in computer science. Using a qualitative approach, Jacques demonstrated how these tools could enhance students' understanding of coding concepts and critical thinking skills by providing multiple solutions to problems, thereby increasing their engagement with the programming language. The work concluded that LLM-based coding tools could serve as valuable resources in improving programmingStudents' experiences of using ChatGPT in an undergraduate programming course education. Daun and Brings [22] identified both potential benefits and drawbacks of using generative AIs like ChatGPT in software engineering education, noting that while ChatGPT performed well in providing answers to software engineering questions and literature references, its unsupervised use could be detrimental. They recommended supervised integration of these tools in educational practices to leverage their benefits while mitigating potential risks.

3. Methodology

We conducted our study with two separate groups of part-time undergraduate students in an introductory Java programming courses. Each group's course lasted five weeks and followed a Python programming course. The students, who had no prior experience with Java, were expected to attend on-campus lectures, complete programming exercises at home, and pass a written exam at the end of the course. The programming exercises, which were identical in both groups, covered a wide range of Java programming concepts, including fundamentals, loops, object-oriented programming, interfaces, collections, file handling, lambda expressions, and multithreading. Both courses also emphasized the importance of code quality, i.e., the adherance to coding conventions, by encompassing regular training sessions designed to instruct students on avoiding code smells. The lecturer provided individualized feedback and graded the students' programming exercises, which were submitted on a weekly basis through the university's online teaching system. Although not mandatory, submission of programming exercises was strongly encouraged to enable students to benefit from the lecturer's feedback.

The first group of student (n = 16, control group) attended the course during the 2022 summer term, thus before the public availability of ChatGPT. The second group of students (n = 22, treatment group) attended it during the 2023 summer term. In this course, students could choose for each programming exercise whether they wanted to use ChatGPT. The decision to use or not use ChatGPT needed to be given during submission of their programming exercise. Table 1 shows the number of exercise submissions among the two groups.

 doi:10.20944/preprints202406.1151.v1

Table 1. Students' submissions of programming exercises.

#	Exercise Focus	Control Group	Treatment Group	
			●	○
Fu	Fundamentals	16	19	3
Lo	Loops	15	19	3
OO	Object-Oriented Programming	15	19	3
In	Interfaces, Exception Handling	15	19	3
Co	Collections	15	15	6
Fi	File, IO Streams	14	16	3
La	Lambda Expressions, Multithreading	13	12	6

● Students using ChatGPT, ○ Students not using ChatGPT

3.1. Static Code Analysis

We used *Checkstyle* [8] to perform static analysis of the source code submitted by students and employed a predefined ruleset [24,25] to detect violations of "Oracle's Code Conventions for Java" [10]. The ruleset encompasses rules specifying the correct implementation of established coding conventions. These data were utilized to answer the first two research questions (RQ1 and RQ2).

Furthermore, to address the third research question (RQ3), we also calculated the cyclomatic and cognitive complexity of the submitted exercise source code. *Cyclomatic complexity* [26] is a quantitative measure that assesses the number of linearly independent paths through a program's source code, reflecting its complexity and potential for modification. It is often employed to predict a program's maintainability, with higher values indicating more intricate code that could be harder to understand and modify. We determined this metric using *Checkstyle*.

Similarly, *cognitive complexity* [27] is a quantitative measure that evaluates the ease of comprehending code, diverging from cyclomatic complexity by focusing on the mental effort required to comprehend code rather than just control flow complexity. Its calculation involves evaluating elements such as the depth of nesting in control structures, the complexity added by logical operators, and the presence of multiple conditions within statements, all contributing to the overall score. This metric was determined using the *SonarQube* platform [28] for static code analysis.

3.2. Statistical Tests

Upon an initial assessment of the data, we discovered that the distribution of rule violations and complexity measures was right-skewed, which suggested a concentration of lower values with a long tail extending towards higher values. This skewness indicated that a considerable number of students had fewer violations, while a smaller subset had a higher number of violations. As the distribution of the data was non-normal, traditional parametric tests, which presume normality of the data distribution, were deemed inappropriate for statistical analysis of the data. Parametric tests rely on the assumption that the data are normally distributed within each group being compared. However, the right-skewed nature of our data violated this assumption, which could lead to erroneous conclusions if such tests were applied.

To address this issue, we opted for the *Wilcoxon Rank Sum Test* with continuity correction for our comparative analysis between the two groups. This non-parametric test does not assume normal distribution of the data and is particularly well-suited for analyzing differences in median values between two independent groups when the data are skewed. It evaluates whether the distribution of rule violations or complexity measures significantly differs between students who used ChatGPT for the course' programming exercises (treatment group) and those who did not (control group). The statistical significance was set to $p \leq 0.005$ and we formulated the following null hypotheses $H_{1\text{-}2,0}$ for research questions RQ1 and RQ2:

There is no correlation between the usage of ChatGPT by students to implement programming exercises and

 doi:10.20944/preprints202406.1151.v1

- $H_{1,0}$: *the number of rule violations (to code coventions).*
- $H_{2.1,0}$: *the cyclomatic complexity of the source code.*
- $H_{2.2,0}$: *the cognitive complexity of the source code.*

As a result, the corresponding alternative hypotheses $H_{1\text{-}2,a}$ can be accepted with a *p-value* of less than 0.05, indicating that there is a statistically significant correlation between the usage of ChatGPT by students to implement programming exercises and rule violations ($H_{1,a}$), and cyclomatic or cognitive code complexity ($H_{2.1,a}$ and $H_{2.2,a}$ respectively).

4. Results

The following section outlines the findings of our study in sequential manner with the research questions.

4.1. Most Frequent Rule Violations (RQ1)

In our initial analysis, we assessed the quality and distribution of the acquired rule violations and complexity measures. Subsequently, we normalized the data to account for the varying number of students in the control and treatment groups, which was essential for maintaining the integrity of our study. After preparing the data, we calculated the key statistical properties for both groups. We focused on assessing the likelihood of observing particular rule violations under the null hypothesis $H_{1,0}$ (see Section 3), which assumes no influence of ChatGPT on the probability of specific rule violations. If the *p-value* of a rule violation is smaller than or equal 0.005, the null hypothesis $H_{1,0}$ for this rule can be rejected, and the alternative hypothesis $H_{1,a}$ can be embraced. This hypothesis signifies that the use of ChatGPT has an effect on the likelihood of violations of this rule.

We ranked the rules in ascending order of their *p-value*, i.e., with decreasing statistical significance. To focus our discussion, we concentrate on the top 12 rule violations with the highest significance. The results from these calculations are presented in Table 3. An asterisk (*) in the first column of the table marks rule violations that have been identified as statistically relevant, i.e., for which the alternative hypothesis $H_{1,a}$ is applicable. Figure 1 shows the rule violations for each programming exercise among the two groups. The definition and rationale of each rule is given in Table 2. In the following, we present the results of our statistical analysis of the students' rule violations.

- **LineLength**: Statistical analysis revealed a significant difference between control and treatment groups, supporting the alternative hypothesis $H_{1,a}$ with a *p-value* of 2.16e-15. The treatment group demonstrated greater adherence this rule across all programming exercises.
- **FinalParameters**: The statistical test supports the alternative hypothesis $H_{1,a}$ with a *p-value* of 9.99e-11, showing significant differences between control and treatment groups. The data reveals consistently higher median violations in the control group across all exercises, emphasizing the improved adherence to this rule when using ChatGPT. For instance, in the *"Object-Oriented Programming"*exercise, the treatment group's median violations are almost half those of the control group.
- **HiddenField**: The statistical analysis revealed a significant difference between the control and treatment groups, with a *p-value* of 1.34e-07, supporting the alternative hypothesis $H_{1,a}$. The treatment group had consistently lower median rule violations across all exercises. As an example, the *"Collections"* exercise had a median of 8.57 violations compared to the control group's median of 16.
- **MissingSwitchDefault**: We found a statistically significant difference with a *p-value* of 8.34e-05, indicating that ChatGPT effectively minimized these violations, thus supporting the alternative hypothesis $H_{1,a}$. The treatment group generally showed lower median violations compared to the control group in all areas. In the *"Interfaces and Exception Handling"* exercise, for example, the median violations for the treatment group were 1.82 and thus significantly lower than the control group's median of 4.

 doi:10.20944/preprints202406.1151.v1

- **DesignForExtension**: The *p-value* of 6.14e-04 confirmed significant differences among the groups, supporting the alternative hypothesis $H_{1,a}$. The median violations consistently favored the treatment group over the control group across all exercises. For example, in the *"Interfaces and Exception Handling"* exercise, the treatment group's median violation was 20, significantly lower than the control group's 34.
- **MagicNumber**: The analysis shows significant differences with a *p-value* of 7.32e-04, supporting the alternative hypothesis $H_{1,a}$. The treatment group generally has lower median violations, such as in the *"Interfaces and Exception Handling"*exercise where it was 7.27 for the treatment group and 15.7 for the control group.
- **VisibilityModifier**: The analysis supports the alternative hypothesis $H_{1,a}$ with a *p-value* of 1.31e-03, indicating significant differences. The treatment group showed greater adherence to this rule, with lower median violations across exercises. For example, in the *"Object-Oriented Programming"* exercise, the median for the treatment group was 1.36, significantly lower than the 4.67 observed in the control group.
- **RightCurly**: Statistical analysis showed a significant variance with a *p-value* of 9.45e-03, supporting the alternative hypothesis $H_{1,a}$ for the comparison between control and treatment groups. The treatment group had consistently lower median violations across different parts of the exercise, indicating better compliance with the rule. For instance, in the *"Interfaces and Exception Handling"* exercise, the median for the treatment group was 1.36, compared to 4.33 for the control group.
- **NeedBraces**: The statistical analysis produced a *p-value* of 0.0100, which suggests a significant difference and supports the alternative hypothesis $H_{1,a}$. The results varied across exercises, with the treatment group often displaying improved compliance by having lower median violations. Specifically, in the *"Interfaces and Exception Handling"* exercise, the treatment group had a median of 1.36, whereas the control group had 13.3, showing a significant reduction in violations. However, in the *"Object-Oriented Programming"* exercise, the treatment group had higher median violations.
- **LocalVariableName**: The statistical results indicate a significant difference with a *p-value* of 0.0116, supporting the alternative hypothesis $H_{1,a}$. The treatment group generally exhibited better adherence to naming conventions for local variables. For instance, in the *"Object-Oriented Programming"* exercise the median was 0.909 for the treatment group compared to 3.33 for the control group.
- **MethodParamPad**: The median violations in both groups were comparable across the exercises, with only minor differences in specific contexts. For example, the treatment group showed a lower median of 1 in the *"File IO and Streams"* exercise compared to the control group's 3.93. However, since the overall *p-value* is 0.057, these differences are not statistically significant at the chosen threshold, and the null hypothesis $H_{1,0}$ can be accepted.
- **AvoidNestedBlocks**: With a *p-value* of 0.154 the difference in this rule's violations between the two groups does not reach statistical significance. Hence the null hypothesis $H_{1,0}$ can be supported. Despite some variations in median violations across different parts of the exercise, such as the treatment group exhibiting a median of 5.68 in the *"Interfaces and Exception Handling"* exercise compared to 15 for the control group, these differences are not statistically robust across all exercises.

 doi:10.20944/preprints202406.1151.v1

Table 2. Definitions of the checked coding rules, adapted from [23].

Rule	Definition
LineLength	Checks for long code lines.
FinalParameters	Checks that parameters for methods, constructors, catch and for-each blocks are marked `final`.
HiddenField	Checks that a local variable or a parameter does not shadow a field that is defined in the same class.
MissingSwitchDefault	Checks that each `switch` statement has a `default` clause.
DesignForExtension	Ensures that classes are structured in a way that facilitates their extension through subclassing.
MagicNumber	Checks that there are no "magic numbers", i.e., a numeric literal not defined as a constant.
VisibilityModifier	Enforces encapsulation by requiring that public class members must be either `static final`, immutable or annotated by specific annotations.
RightCurly	Checks the placement of right curly braces ('}') for code blocks.
NeedBraces	Checks for braces around code blocks.
LocalVariableName	Checks that local, non-final variable names conform to Java naming conventions.
MethodParamPad	Evaluates the spacing between method and constructor definitions, calls, and invocations, and the opening parentheses of parameter lists.
AvoidNestedBlocks	Identifies instances where blocks of code are nested within other blocks.

Table 3. Standard distribution (σ), median (μ), mean ($\bar{x}$) and *p-values* for $H_{1,0}$ (rule violations $\sim$ ChatGPT usage).

			Control Group			Treatment Group		
	Rule	**p-value**	σ	μ	$\bar{x}$	σ	μ	$\bar{x}$
*	LineLength	< 0.005	32.71	50.3	54.36	15.91	16.65	21.18
*	FinalParameters	< 0.005	19.22	28.57	33.24	10.62	16.23	17.13
*	HiddenField	< 0.005	6.24	9.91	11.61	3.96	6.32	6.81
*	MissingSwitchDefault	< 0.005	2.02	2.80	3.13	1.43	0.92	1.59
*	DesignForExtension	< 0.005	14.95	23.51	25.19	10.94	18.10	17.54
*	MagicNumber	< 0.005	9.48	15.42	16.28	7.46	11.42	12.05
*	VisibilityModifier	< 0.005	2.20	2.97	3.49	2.02	1.69	2.28
*	RightCurly	< 0.005	3.92	2.74	4.05	2.81	1.55	2.59
*	NeedBraces	< 0.005	7.01	5.37	7.99	3.91	2.69	3.83
*	LocalVariableName	< 0.005	1.88	2.37	2.90	1.33	1.03	1.60
	MethodParamPad	0.057	3.07	2.66	3.79	2.27	1.31	2.41
	AvoidNestedBlocks	0.154	7.49	8.07	10.76	5.19	5.57	6.85
	Complexity Measure							
*	CyclomaticComplexity	< 0.005	8.67	9.17	12.2	4.80	7.27	8.82
*	CognitiveComplexity	< 0.005	10.4	1.50	3.30	6.78	1.34	2.88

 doi:10.20944/preprints202406.1151.v1

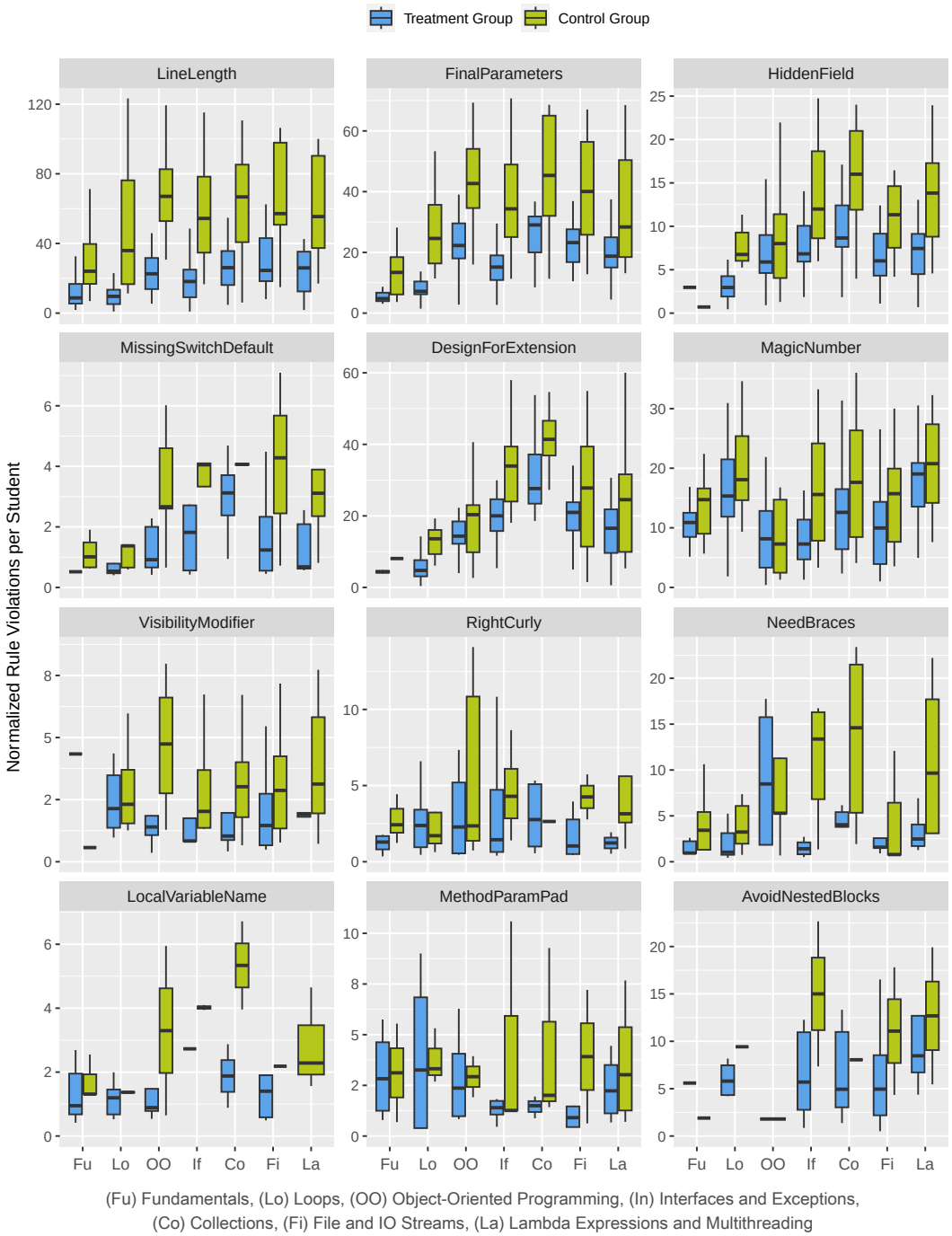

Figure 1. Most frequent rule violations accross the different programming exercises. The treatment group (blue) shows students using ChatGPT, the control group (green) students not using ChatGPT.

Given the limited sample size, these results should be interpreted cautiously. To delve deeper into the underlying factors contributing to these disparities, additional research with larger sample sizes is necessary.

4.2. Most Violation-Dense Topics (RQ2)

Figure 2 shows the total number of rule violations per exercise and group. For the sake of brevity, we only present the top three exercises with the highest frequency of rule violations and the leading five rules that were violated most frequently across all exercises for each group. We identified *"Collections"*,

 doi:10.20944/preprints202406.1151.v1

"File IO and Streams", and *"Object-Oriented Programming"* as the exercises with the highest number of rule violations for both groups.

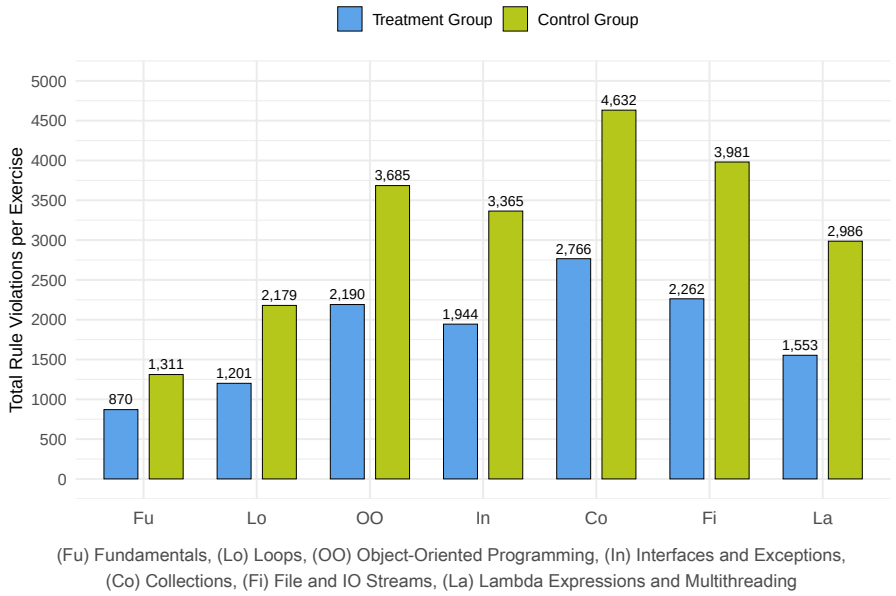

Figure 2. Distribution of rule violations accross the different programming exercises. The treatment group (blue) shows students using ChatGPT, the control group (green) students not using ChatGPT.

For the control group, there were 4,632 violations of the *"Collections"* exercise, 3,981 violations of the *"File IO and Streams"* exercise, and 3,685 violations of the *"Object-Oriented Programming"* exercise among a total of 22,139 rule violations across all exercises. Among violations of other rules, 6,903 violations relate to the *LineLength* rule, 3,544 violations to the *FinalParameters* rule, 2,223 violations to the *DesignForExtension* rule, 1,566 violations to the *MagicNumber* rule, and 1,107 violations to the *HiddenField* rule.

In contrast, for the treatment group, there were 2,766 violations of the *"Collections"* exercise, 2,262 violations of the *"File IO and Streams"* exercise, and 2,190 violations of the *"Object-Oriented Programming"* exercise among a total of 12,786 rule violations across all exercises. Among violations of other rules, 2,686 violations relate to the *LineLength* rule, 2,168 violations to the *FinalParameters* rule, 1,793 violations to the *DesignForExtension* rule, 1,490 violations to the *MagicNumber* rule, and 699 violations to the *HiddenField* rule.

4.3. Influence on Cyclomatic and Cognitive Complexity (RQ3)

Figure 3 shows the distribution of cyclomatic and cognitive complexity of the students' implementation of the programming exercises. The statistical analysis concerning the cyclomatic complexity of the students' code showed a *p-value* of 7.91e-14, indicating a significant difference and thus supporting the alternative hypothesis $H_{3.1,a}$. This notable discrepancy underscores a trend to lower cyclomatic complexity of the code in the treatment group, evident across various parts of the exercise. For instance, in *"Object-Oriented Programming"* exercise, the treatment group reported a median complexity of 5.91, substantially lower than the control group's 8.67. Such patterns are consistent across all exercises, with the treatment group frequently demonstrating lower median complexity measures. This suggests that ChatGPT potentially aids in producing code with reduced cyclomatic complexity.

 doi:10.20944/preprints202406.1151.v1

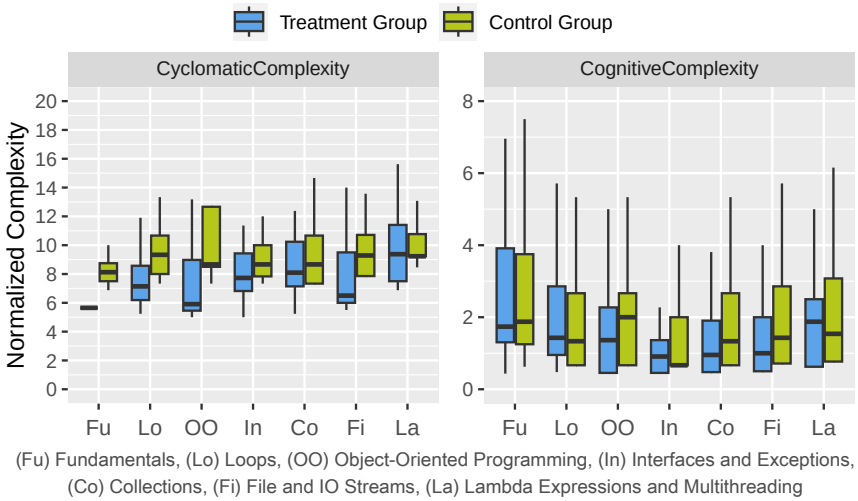

Figure 3. Complexity measures of students' code when using (treatment group) or not using ChatGPT for implementing programming exercises (control group).

Similarly, the statistical analysis of the cognitive complexity revealed a *p-value* of 6.45e-22, thus supporting the alternative hypothesis $H_{3.2,a}$. This significant finding indicates lower cognitive complexity in the treatment group across various programming exercises. For instance, in the *"Object-Oriented Programming"* exercise, the treatment group's median complexity was 1.36, compared to 2.00 in the control group. This trend of lower median complexity values in the treatment group suggests that ChatGPT slightly assists students in writing code that is easier to understand. However, given the only small difference in the median values we regard this effect as rather negligible and requiring further research with larger student populations for further evaluation.

5. Discussion

The results of our study indicate a consistent pattern, wherein the treatment group that utilized ChatGPT displayed significantly fewer violations across several Java programming rules compared to the control group. This pattern was particularly evident in rules pertaining to code structure and syntax, such as *LineLength*, *FinalParameters*, and *MissingSwitchDefault*.

One of the most noteworthy observations was the treatment group's ability to more closely adhere to Java programming best practices, as evidenced by their lower median violations in rules like *DesignForExtension* and *VisibilityModifier*. These results suggest that incorporating ChatGPT into programming education can not only assist students in writing syntactically correct code but also in understanding and applying good design principles, which are essential for creating maintainable and scalable software.

Furthermore, the reduction in cyclomatic and cognitive complexities in the treatment group's code underscores another vital educational benefit. Lower complexity measures often correlate with simpler, more readable code, which not only reduces the cognitive load on programmers but also potentially decreases the likelihood of bugs and errors. This finding is crucial for educators as it emphasizes the potential of tools like ChatGPT to enhance students' ability to write efficient, understandable, and less complex code, thereby improving their overall coding proficiency.

Our research further revealed that violations to special coding rules, including *MethodParamPad* and *AvoidNestedBlocks*, did not result in statistically significant differences between the groups. This suggests that while ChatGPT can be beneficial for many coding aspects, there may be nuances in coding style and structure that may not be adequately addressed if students rely solely on assistance from an LLM like ChatGPT. Hence, educators must remain cognizant of the limitations and ensure that students also develop the critical thinking and problem-solving skills necessary for effective programming independent of such tools.

 doi:10.20944/preprints202406.1151.v1

6. Threats to Validity

We acknowledge several potential threats to the validity of our study that could impact the interpretation and generalizability of our findings.

Regarding **internal validity**, one major concern is the level of experience of the participants. Given that the students were novices with no prior experience in Java programming, their learning curve over the course could affect their ability to adhere to coding conventions and manage code complexity, regardless of the assistance provided by ChatGPT. Additionally, the individualized feedback provided by the lecturer may have varied in its effectiveness across students, potentially confounding results related to improvements in code quality.

In terms of **external validity**, the small sample size poses a significant threat, limiting the generalizability of the results to a broader population of programming students. Moreover, the specific educational context of part-time undergraduate students may not represent the diverse backgrounds and educational settings in which programming is taught, affecting the applicability of the findings to other learning environments.

The measurement of code quality through static code analysis tools and predefined rulesets to detect violations may raise concerns about **construct validity**. The reliance on these tools and rulesets to quantify code quality and complexity may not fully capture the nuances of what constitutes high-quality, maintainable code, potentially overlooking aspects of code quality not encompassed by static code analysis per se.

Similarly, concerns have been expressed about **statistical conclusion validity** of the non-parametric *Wilcoxon Rank Sum Test*, which is often used in non-normal distributions. Given the right-skewed distribution of rule violations and complexity measures, there is a risk that the test may not have the power to detect subtle differences between groups due to outliers or the distribution shape.

7. Conclusion

Our research systematically investigated the impact of ChatGPT on code quality among undergraduate students through a detailed analysis of rule violations and complexity metrics. Subsequently, we present the key findings in alignment with our research questions.

- **Most Frequent Rule Violations:** Our investigation revealed statistically significant differences in rule adherence between students using ChatGPT (treatment group) and those not using it (control group). Specifically, the treatment group showed fewer violations across several key coding conventions. Notably, rules verifying the length of code lines (*LineLength*), the declaration of parameters for methods, constructors, catch blocks, and for-each loops as `final` (*FinalParameters*) or verifying code extensibility (*DesignForExtension*) showed marked improvements, with *p-values* less than 0.005, indicating a statistically significant difference in adherence between the two groups.
- **Most Violation-Dense Topics**: We observed exercises related to *"Collections"*, *"File IO and Streams"*, and *"Object-Oriented Programming"* as topics with the highest incidence of rule violations. Despite a significant overall reduction in violations among the treatment group, these areas remained challenging, underscoring the need for targeted educational interventions. The rules *LineLength* and *FinalParameters* were among the most frequently violated, highlighting specific areas where ChatGPT's guidance notably improved student performance.
- **Influence on Cyclomatic and Cognitive Complexity:** Our study further explored the cyclomatic and cognitive complexity of student submissions. We observed that code from the treatment group displayed lower complexity levels across both metrics, notably in cyclomatic complexity (p = 7.91e-14). These findings imply that ChatGPT contributes not merely to simplifying code complexity but also to enhancing code comprehensibility, as evidenced by the statistically significant difference in cognitive complexity (p = 6.45e-22).

The outcomes of our study offer novel insights into the impact of ChatGPT on code quality among novice programming students. Nevertheless, it is crucial to exercise caution when interpreting the

 doi:10.20944/preprints202406.1151.v1

results, taking into account the identified threats to validity. Expanding on these findings requires future research involving larger, more diverse samples, considering various educational contexts and more refined measures of code quality and ChatGPT usage. In addition, it is essential to explore the experiences of a larger number of students, possibly also assessing ChatGPT's suitability for supporting students in programming courses beyond Java. A qualitative analysis of the students' prompts to ChatGPT could help examine the structure and quality of their input, enabling educators to better instruct students on how to tailor their input to ChatGPT to improve code quality.

Author Contributions: Conceptualization: P.H. and G.W.; methodology, P.H. and G.W.; software, P.H..; validation, P.H.; formal analysis, P.H.; investigation, P.H. and G.W.; resources, G.W.; data curation, P.H.; writing—original draft preparation, P.H.; writing—review and editing, P.H.; visualization, P.H.; All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Informed Consent Statement: Informed consent was obtained from all subjects involved in the study.

Data Availability Statement: Restrictions apply to the datasets due to privacy and confidentiality agreements related to student-related performance data.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. OpenAI. OpenAI. https://openai.com/, 2024. Accessed: 2024-06-14.
2. Ozkaya, I. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software* **2023**, *40*, 4–8. doi:10.1109/MS.2023.3248401.
3. Breuker, D.M.; Derriks, J.; Brunekreef, J. Measuring static quality of student code. Proceedings of the 16th annual joint conference on Innovation and technology in computer science education; Association for Computing Machinery: New York, NY, USA, 2011; ITiCSE '11, pp. 13–17. doi:10.1145/1999747.1999754.
4. Izu, C.; Mirolo, C. Exploring CS1 Student's Notions of Code Quality. Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1; Association for Computing Machinery: New York, NY, USA, 2023; ITiCSE 2023, pp. 12–18. doi:10.1145/3587102.3588808.
5. Östlund, L.; Wicklund, N.; Glassey, R. It's Never too Early to Learn About Code Quality: A Longitudinal Study of Code Quality in First-year Computer Science Students. Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1; Association for Computing Machinery: New York, NY, USA, 2023; SIGCSE 2023, pp. 792–798. doi:10.1145/3545945.3569829.
6. Brown, N.C.; Altadmri, A. Investigating novice programming mistakes: educator beliefs vs. student data. Proceedings of the tenth annual conference on International computing education research; Association for Computing Machinery: New York, NY, USA, 2014; ICER '14, pp. 43–50. doi:10.1145/2632320.2632343.
7. De Ruvo, G.; Tempero, E.; Luxton-Reilly, A.; Rowe, G.B.; Giacaman, N. Understanding semantic style by analysing student code. Proceedings of the 20th Australasian Computing Education Conference; Association for Computing Machinery: New York, NY, USA, 2018; ACE '18, pp. 73–82. doi:10.1145/3160489.3160500.
8. checkstyle. checkstyle – Checkstyle 10.14.1. https://checkstyle.sourceforge.io/, 2024. Accessed: 2024-06-14.
9. PMD. PMD. https://pmd.github.io/, 2024. Accessed: 2024-06-14.
10. Oracle. Code Conventions for the Java Programming Language: Contents. https://www.oracle.com/java/technologies/javase/codeconventions-contents.html, 2024. Accessed: 2024-06-14.
11. Li, K.; Hong, S.; Fu, C.; Zhang, Y.; Liu, M. Discriminating Human-authored from ChatGPT-Generated Code Via Discernable Feature Analysis. 2023 IEEE 34th International Symposium on Software Reliability Engineering Workshops (ISSREW), 2023, pp. 120–127. doi:10.1109/ISSREW60843.2023.00059.
12. Guo, Q.; Cao, J.; Xie, X.; Liu, S.; Li, X.; Chen, B.; Peng, X. Exploring the Potential of ChatGPT in Automated Code Refinement: An Empirical Study. Proceedings of the 46th IEEE/ACM International Conference on Software Engineering; Association for Computing Machinery: New York, NY, USA, 2024; ICSE '24, pp. 1–13. doi:10.1145/3597503.3623306.

13. Liu, Y.; Le-Cong, T.; Widyasari, R.; Tantithamthavorn, C.; Li, L.; Le, X.B.D.; Lo, D. Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues. *ACM Transactions on Software Engineering and Methodology* **2024**. Just Accepted, doi:10.1145/3643674.
14. Börstler, J.; Störrle, H.; Toll, D.; van Assema, J.; Duran, R.; Hooshangi, S.; Jeuring, J.; Keuning, H.; Kleiner, C.; MacKellar, B. "I know it when I see it" Perceptions of Code Quality: ITiCSE '17 Working Group Report. Proceedings of the 2017 ITiCSE Conference on Working Group Reports; Association for Computing Machinery: New York, NY, USA, 2018; ITiCSE-WGR '17, pp. 70–85. doi:10.1145/3174781.3174785.
15. Brown, N.C.C.; Weill-Tessier, P.; Sekula, M.; Costache, A.L.; Kölling, M. Novice Use of the Java Programming Language. *ACM Transactions on Computing Education* **2022**, *23*, 10:1–10:24. doi:10.1145/3551393.
16. Karnalim, O.; Simon.; Chivers, W. Work-In-Progress: Code Quality Issues of Computing Undergraduates. 2022 IEEE Global Engineering Education Conference (EDUCON), 2022, pp. 1734–1736. ISSN: 2165-9567, doi:10.1109/EDUCON52537.2022.9766807.
17. Keuning, H.; Heeren, B.; Jeuring, J. Code Quality Issues in Student Programs. Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education; Association for Computing Machinery: New York, NY, USA, 2017; ITiCSE '17, pp. 110–115. doi:10.1145/3059009.3059061.
18. Keuning, H.; Jeuring, J.; Heeren, B. A Systematic Mapping Study of Code Quality in Education. Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1; Association for Computing Machinery: New York, NY, USA, 2023; ITiCSE 2023, pp. 5–11. doi:10.1145/3587102.3588777.
19. Brown, N.C.C.; Altadmri, A. Novice Java Programming Mistakes: Large-Scale Data vs. Educator Beliefs. *ACM Transactions on Computing Education* **2017**, *17*, 7:1–7:21. doi:10.1145/2994154.
20. Kazemitabaar, M.; Chow, J.; Ma, C.K.T.; Ericson, B.J.; Weintrop, D.; Grossman, T. Studying the effect of AI Code Generators on Supporting Novice Learners in Introductory Programming. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems; Schmidt, A.; Väänänen, K.; Goyal, T.; Kristensson, P.O.; Peters, A.; Mueller, S.; Williamson, J., Eds.; Association for Computing Machinery: New York, NY, USA, 2023; CHI '23, pp. 1–23. doi:10.1145/3544548.3580919.
21. Jacques, L. Teaching CS-101 at the Dawn of ChatGPT. *ACM Inroads* **2023**, *14*, 40–46. doi:10.1145/3595634.
22. Daun, M.; Brings, J. How ChatGPT Will Change Software Engineering Education. Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1; Laakso, M.J.; Monga, M., Eds.; Association for Computing Machinery: New York, NY, USA, 2023; ITiCSE 2023, pp. 110–116. doi:10.1145/3587102.3588815.
23. checkstyle. Checks. https://checkstyle.sourceforge.io/checks.html, 2024. Accessed: 2024-06-14.
24. checkstyle. Sun's Java Style. https://checkstyle.sourceforge.io/sun_style.html, 2024. Accessed: 2024-06-14.
25. checkstyle. checkstyle/checkstyle. https://github.com/checkstyle/checkstyle/blob/master/src/main/resources/sun_checks.xml, 2024. Accessed: 2024-06-14.
26. McCabe, T. A Complexity Measure. *IEEE Transactions on Software Engineering* **1976**, *SE-2*, 308–320. Conference Name: IEEE Transactions on Software Engineering, doi:10.1109/TSE.1976.233837.
27. Campbell, G.A. Cognitive complexity: an overview and evaluation. Proceedings of the 2018 International Conference on Technical Debt; Association for Computing Machinery: New York, NY, USA, 2018; TechDebt '18, pp. 57–58. doi:10.1145/3194164.3194186.
28. SonarSource. Code Quality Tool & Secure Analysis with SonarQube. https://www.sonarsource.com/products/sonarqube/, 2024. Accessed: 2024-06-14.