IIAI Open Conference Publication Series
IIAI Letters on Institutional Research
Vol.004, LIR293
DOI: https://doi.org/10.52731/lir.v004.293

Quantitative Quality Evaluation of the Impact of Indentation in Source Code Using Eye-Tracking

Kou Yorimoto [*], Shimpei Matsumoto [*]

Abstract

This study focuses on the setting of indentation and aims to elucidate its impact on readability through the analysis of program comprehension processes using eye tracking. Within the workload of software lifecycle activities, maintenance tasks are known to occupy a significant proportion. Among the various stages of maintenance, understanding the content of source code, namely comprehension, is considered the most time-consuming task. Against this backdrop, the ability to comprehend source code is recognized as an important programming skill. Alongside comprehension, awareness of source code readability is also considered a vital aspect of comprehension learning. Factors influencing the readability of source code include code structure, naming conventions, presence, and quality of comments, as well as indentation and placement of parentheses. However, insufficient quantitative research has been conducted to demonstrate the impact of these factors on readability. Therefore, this study focuses on the influence of indentation on readability and analyzes the program comprehension process using eye tracking. The results suggest that in the case of small-scale source code, the absence of indentation may not adversely affect slicing.

Keywords: Eye tracking, source code, indentation, readability

1 Background/Objectives and Goals

In the software lifecycle, maintenance tasks constitute a remarkably high proportion, ranging from 40% to 67% [1]. Indeed, throughout the process from software development to operation and its eventual discontinuation, maintenance tasks are the most costly and time-consuming stages [2]. Among all maintenance processes, understanding the content of the source code (comprehension or reading) is considered the most time-intensive task.

Against this backdrop, the ability to comprehend source code is regarded as a crucial programming skill. In recent years, as software development has evolved, the impact of code readability on the overall success of projects has become increasingly significant, especially when teams tackle complex projects. Improving comprehension not only facilitates bug discovery and correction, as well as maintaining software performance and stability, but also directly contributes to the efficiency of development tasks such as system enhancement and addition of new features. Furthermore, source code comprehension plays a vital role during peer reviews. Thus, education aimed at enhancing source code comprehension has long been practiced in higher education institutions.

Recent studies have investigated various aspects of source code comprehension learning and its effectiveness. For instance, researchers have reported that providing annotated source code or visually representing the structure of source code contributes to improved

[*] Hiroshima Institute of Technology, Hiroshima, Japan

comprehension [3][4][5]. Open-source tools for visualizing source code structure have also been released and widely used in educational and development settings.

In the study by Hanabusa et al., experimental tasks were based on source code consisting of only three lines of assignment statements, generating four patterns of data dependencies. As a result, the researchers elucidated the characteristics of gaze transitions between lines in the source code and confirmed the influence of each pattern of data dependencies.

This revealed that learners comprehend source code using data dependencies as clues, enabling judgment on whether learners appropriately understand the program's dependencies when their reading deviates from data dependencies. Although Hanabusa et al.'s findings are fundamental for evaluating learners' proficiency and the quality of comprehension materials, the targeted comprehension tasks were very simplistic. Since only source code with four patterns of data dependencies generated from three lines of assignment statements was used as experimental tasks, the insights gained are considered limited. Therefore, Hiratani conducted experiments focusing on source code comprehension involving control syntax using the same method as Hanabusa et al. [6]. As a result, Hiratani clearly demonstrated that gaze transitions at focal points are indeed influenced by the structure of the program.

Focusing attention on source code readability is also considered essential for effective comprehension learning. Factors influencing source code readability include code structure, naming conventions, presence and quality of comments, indentation settings, and positioning of braces. Hence, this study focuses on settings indentation, analyzing the process of program comprehension using eye-tracking to elucidate the impact of these settings on readability. Hereinafter, the factors affecting source code readability as mentioned will be referred to as coding conventions in this study.

The background/objectives and goals section provide a comprehensive overview of the research context and objectives, highlighting the significance of maintenance tasks in the software lifecycle and the importance of source code comprehension as a crucial programming skill. The authors emphasize the impact of code readability on the overall success of projects, especially when teams tackle complex projects, and discuss the various benefits of improving comprehension, such as facilitating bug discovery and correction, maintaining software performance and stability, and contributing to the efficiency of development tasks.

The authors also discuss the role of source code comprehension in peer reviews and the long-standing practice of education aimed at enhancing source code comprehension in higher education institutions. They review recent studies investigating various aspects of source code comprehension learning and its effectiveness, such as providing annotated source code or visually representing the structure of source code to improve comprehension, and the release and widespread use of open-source tools for visualizing source code structure in educational and development settings.

Overall, the background/objectives and goals section effectively sets the stage for the study, providing a clear and comprehensive overview of the research context, objectives, and methodology. The authors successfully communicate the significance of the study in addressing the importance of source code comprehension and readability in the software lifecycle and the potential of eye-tracking as a tool to analyze the impact of indentation settings on readability. The section is well-structured and engaging, effectively guiding the reader into the subsequent sections of the paper.

2 Methods

In this study, the focus is on key elements of source code materials, namely indentation. The objective is to elucidate the factors influencing comprehension accuracy caused by these settings using eye-tracking. This chapter explains various concepts related to the elements crucial for the analysis methodology of this study.

In this study, comprehension accuracy is defined as the accuracy of slicing. When given a task to determine the values of variables after processing a certain source code, the lines necessary to solve that task are clearly identifiable through backward slicing. By utilizing this characteristic, quantifying the extent to which unnecessary lines are viewed to solve the task allows the calculation of comprehension efficiency. Henceforth, this comprehension efficiency will be referred to as comprehension accuracy, and the unnecessary lines needed to solve the task will be termed as “dummy lines”.

Assessing the degree of attention to unnecessary lines can be partially evaluated by the accuracy of comprehension materials and interviews with learners. However, elucidating the reasons for learners' evaluations and why the accuracy rates were high or low is not straightforward. Eye-tracking is useful in this regard. Conducting eye-tracking enables us to determine which parts of the source code learners are viewing with confusion or taking time to understand. As an analytical method for the data obtained from eye-tracking, the transition probabilities to each node are calculated using a simple Markov model. One of the data acquisition methods employed in this study was based on the transition probability analysis technique used by Hanabusa et al [7]. The hypothesis is that the more appropriate the indentation used in the code due to the provision of “dummy lines”, the lower the transition probability to the “dummy lines”.

To investigate the influence of data dependencies in source code comprehension, the transition probabilities of a simple Markov process model are employed to analyze learners' eye movements between lines of code. A Markov process refers to a stochastic process characterized by Markovian properties, where future behavior is determined solely by the present state, independent of past behavior. In probability theory, Markov process denotes a type of stochastic process where the conditional probability distribution of future states depends only on the current state and is independent of any previous states. In other words, given the past states, the current state is a conditional probability. Typically, the distribution of a Markov process that appears is determined by transition probabilities. Transition probabilities $P(s,t;x,Y)$of a Markov process x_t refer to the probability of transitioning from a point x in the state space at time s to a (measurable) subset Y of the state space at time $t > s$ and are defined as follows:

$$P(s,t;x,Y) = P(x_t \in Y | x_s = x) \tag{1}$$

Furthermore, Markov processes are classified into several categories, one of which is the simple Markov process. A simple Markov process refers to a Markov process where the future state is determined solely by the current state. Among Markov processes, those with discrete states (finite or countable) are referred to as Markov chains. Markov chains exhibit the property of Markovian independence, where future behavior is determined solely by the current state, independent of past behavior. Regarding state transitions or movements occurring at each time step, Markov chains are sequences determined solely by the current

state, irrespective of past states. Markov chains consist of a sequence of random variables X_1 , X_2 , X_3 , …where given the current state, both past and future states are independent, as defined by the following equation:

$$\Pr(X_{n+1} = x | X_n = x_n, \ldots, X_1 = x_1, X_0 = x_0) = \Pr(X_{n+1} = x | X_n = x_n) \tag{2}$$

The possible values of Xi constitute the state space of the chain, denoted as a countable set S. Markov chains are represented by directed graphs, where edges indicate the probability of transitioning from one state to another. In a temporally homogeneous Markov chain, the process is described by a matrix $\mathrm{p_{ii}}$ that does not depend on time, and the sum of elements π_j in vector π is equal to 1, satisfying the following equation:

$$\pi_j = \sum_{i \in S} \pi_i \, p_{ij} \tag{3}$$

In this case, the vector π is referred to as the stationary distribution. A chain is said to be irreducible if all of its states are recurrent, and only in this case does it possess a stationary distribution. In such instances, π is unique, and there exists the following relationship with the expected value of recurrence time M_j

$$\pi_j = \frac{1}{M_j} \tag{4}$$

If the state space is finite, the transition probability distribution is represented by a matrix known as the transition matrix. The transition matrix describes a Markov chain $\mathrm{X_t}$ over a finite set of states S (cardinality S). When the probability of transitioning from state i to state j in one step is $Pr(j|i) = P_{i,j}$ the probability matrix P is a matrix with i rows and j columns, where $P_{i,j}$represents the element at the intersection of the i-th row and the j-th column.

$$P = \begin{bmatrix} P_{1,1} & P_{1,2} & \cdots & P_{1,j} & \cdots & P_{1,S} \\ P_{2,1} & P_{2,2} & \ldots & P_{2,j} & \cdots & P_{2,S} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ P_{i,1} & P_{i,2} & \cdots & P_{i,j} & \cdots & P_{i,S} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ P_{S,1} & P_{S,2} & \cdots & P_{S,j} & \cdots & P_{S,S} \end{bmatrix} \tag{5}$$

The sum of probabilities of transitioning from state i to the next state equals 1, thus resulting in the following equation:

$$\sum_{j=1}^{S} P_{i,j} = 1 \tag{6}$$

A square matrix with non-negative real components, where the sum of each row equals 1, satisfies the conditions of a right stochastic matrix. Furthermore, if the Markov chain is temporally homogeneous, meaning the transition matrix P does not depend on the index n, then the k -step transition probability can be expressed as the k-th power of the transition matrix, denoted as P^k.

The stationary distribution π is a row vector that satisfies the following equation:

$$\pi = \pi P \tag{7}$$

If P^k converges to a matrix where each row corresponds to the stationary distribution π, then convergence can be expressed as follows:

$$\lim_{k=\infty} P^k = 1\pi \tag{8}$$

In other words, as time progresses, a Markov chain converges to a stationary distribution regardless of the initial distribution. Moreover, the distribution at the limit as $t \to \infty$ is referred to as the limiting distribution. A Markov chain is represented by the following equation:

$$\pi_i p_{i,j} = \pi_j p_{j,i} \tag{9}$$

If such exists, the Markov chain is referred to as reversible. In a reversible Markov chain, π always represents the stationary distribution.

In this study, the portions of code where processing is described are considered as states, and the transitions between these portions are conceptualized as edges, constructing a state transition model that is regarded as a model of learners' thought processes. The authors believe that there exists a Markovian property in the transitions of gaze movements. The focus in this study is solely on the transitions between lines of code, without emphasizing hidden states between the observed output symbol sequences of eye movements and internal cognitive processes. Therefore, in this chapter, a simple model is opted for rather than a hidden Markov model.

Additionally, within the realm of gaze movements, only transitions of attention from one line to another are considered, disregarding transitions to the same state. For instance, representing the transition of gaze between three nodes with a simple Markov process yields the diagram shown in Figure 1. The edge from Node 1 to Node 2 denotes the probability of transitioning from observing the first line to the second line. This transition can be represented by a transition matrix as shown below.

$$\begin{bmatrix} 0 & 0.50 & 0.50 \\ 0.45 & 0 & 0.55 \\ 0.58 & 0.42 & 0 \end{bmatrix} \tag{10}$$

Calculating transition probabilities using the limiting distribution yields probabilities of 0.341 for (1), 0.315 for (2), and 0.344 for (3). In this study, the transition probabilities for each line are calculated using the limiting distribution.

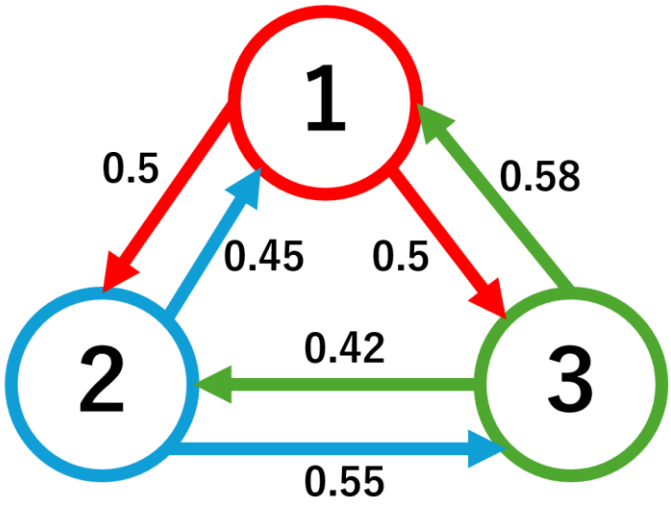

Figure 1: Example of a Simple Markov Model for Eye Movement

Overall, the methods section provides a clear and comprehensive explanation of the concepts and analytical methods used in the study to investigate the impact of indentation settings on source code comprehension using eye-tracking. The authors effectively use mathematical equations and diagrams to illustrate the concepts and provide concrete examples to aid the reader's understanding. The section is well-structured and informative, effectively communicating the methodology used in the study.

3 Test Environment.

Participants consisted of a total of 17 undergraduate and graduate students majoring in informatics who had acquired basic knowledge of C programming and algorithms. Initially, participants' prior knowledge was assessed through a pretest. Based on the results of the pre-test, the subjects were divided into three groups. The groups, ordered from highest to lowest performance, are designated as A, B, and C.

To ensure focus on the task and create a relaxed experimental environment, external auditory distractions were eliminated, and a classroom with no visibility from outside was utilized.

Eye movements of participants were measured using eye-tracking technology while they read and comprehended the displayed programs during the experiment. Participants were required to answer questions presented in accordance with the given tasks, which included tasks to determine the output of the code upon execution, values of variables at specific points, and all elements of arrays. Upon providing answers, participants terminated the eye-tracking measurement by pressing any key and switching screens.

Participants were provided with notepads for their unrestricted use. To measure the time spent viewing the notes and transitions to the notes, an additional measurement area was placed at the bottom of the screen, and participants passed through this area when using the notes. Answers were recorded in the provided notepads at the discretion of the participants.

A total of nine tasks were prepared, comprising three tasks each for codes with regular 4-character indentation, no indentation, and random indentation, respectively. To minimize the influence of habituation, the order of presentation of tasks was varied for each participant, although the content of the codes remained consistent. “Dummy lines”, which were unnecessary for answering the tasks, were included in the displayed code. Failure to focus eye transitions on these “dummy lines” was considered indicative of a lack of comprehension of dependencies.

Eye movement data obtained during the experiment were analyzed by focusing on fixation coordinates and considering transitions between fixations as transitions of interest. Eye movements were measured using the Tobii pro nano eye tracker by Tobii Technology. To mitigate the influence of minor eye movements on transitions, each line of code was spaced apart. Additionally, after completion of each task related to each indentation setting, participants were surveyed on cognitive load, and interviews regarding task difficulty and coding practice settings were conducted.

The code used in this study was limited to small-scale and simple control structures. The only difference between the codes for each task was the indentation settings, ensuring that the difficulty level remained constant.

Regarding the content, participants were required to answer a total of 22 questions, comprising three types of loads: intrinsic task load, extraneous task load, and germane load, with 7 questions for intrinsic task load, 7 questions for extraneous task load, and 8 questions for germane load. Given that the experiment involved three iterations of altering indentation settings, each participant was subjected to 66 questions in total, i.e., 22 questions multiplied by three stages. This sequential survey approach allowed for the examination of differences between stages. To prevent variations in results due to fatigue, eye strain, and presentation order, the sequence of tasks presented varied for each subject.

Specifically, questions pertaining to intrinsic task load focused on the content of the code, such as variables and algorithms involved in reading the source code. Extraneous task load questions addressed factors unrelated to processing, such as the number of lines of code, design, and efficiency in learning. Germane load questions inquired whether reading the source code contributed to skill improvement. The questionnaire was developed with reference to the cognitive load questionnaire used by Hanabusa et al [8].

Furthermore, participants rated their responses on a scale of 0 to 10, with higher numbers indicating greater perceived load.

For intrinsic task load, higher ratings implied that the code content was beneficial for skill enhancement. Conversely, higher ratings for extraneous task load indicated difficulty in comprehending factors other than the code content, while higher ratings for germane load were associated with increased learning effectiveness.

4 Experimental Results and Discussion

The experimental results and discussion section presents the findings of the study and provides a comprehensive analysis and interpretation of the results. The authors begin by explaining the statistical methods used in the study, specifically Welch's t-test for investigat-

ing statistical significance and the use of symbols in graphical representations to denote significance levels.

Additionally, in the graphical representations, symbols were used to denote significance levels, with * indicating $p < .1$, ** indicating $p < .05$, and *** indicating $p < .01$.

The authors then present the obtained results in a series of figures, with Figure 2 depicting the response time, Figure 3 illustrating the percentage of time spent viewing “dummy lines”, Figures 4 presenting the transition probabilities to “dummy lines”, and Figure 5 respectively delineating the intrinsic cognitive load, extraneous cognitive load, and germane cognitive load.

The authors summarize the obtained results, noting a negative relationship between programming proficiency and transition probability to “dummy lines” in both the case of regular 4-character indentation and random indentation. They also report a statistically significant difference ($p < .05$, two-tailed) between evaluations A and C, as revealed by Welch's t-test. Based on these findings, the authors suggest that a more precise explanation, rather than the vague notion of "readability" associated with indentation, could be provided as a consideration to enhance slicing accuracy for individuals less proficient in programming.

The authors also note that no significant differences in programming proficiency were observed in the absence of indentation, and that the tasks in the study involved basic and small-scale code. They suggest that this result indicates the possibility that learners tend to write source code with inappropriate indentation settings because, in the case of small and simple control structure programs, indentation settings have little impact on comprehension efficiency.

The authors then discuss an interesting finding that although the group with a 4-character indentation had the highest score for the probability of transition to dummy lines, the group had the lowest percentage of time spent viewing “dummy lines”. They report that post-experiment interviews revealed that participants intentionally reviewed “dummy lines” multiple times in the 4-character indentation setting to reconfirm the code before answering and suggest that this behavior may be linked to factors such as motivation to recheck the code.

Regarding subjective evaluations conducted after the experiment, the authors report that Welch's t-test indicated significant differences (two-tailed) in extraneous task load and germane load based on indentation settings. They interpret these results as suggesting that participants' subjective perceptions indicate an impact of indentation on aspects such as stress during comprehension.

The authors also note that the observed differences in germane load suggest the importance for instructors to emphasize indentation settings, particularly when incorporating peer review into learning activities.

Overall, the experimental results and discussion section provides a clear and comprehensive analysis and interpretation of the findings of the study. The authors effectively use statistical methods to investigate the significance of the results and provide a thorough discussion of the implications of the findings for programming education and practice. The section is well-structured and informative, effectively communicating the key findings of the study and their implications for the field.

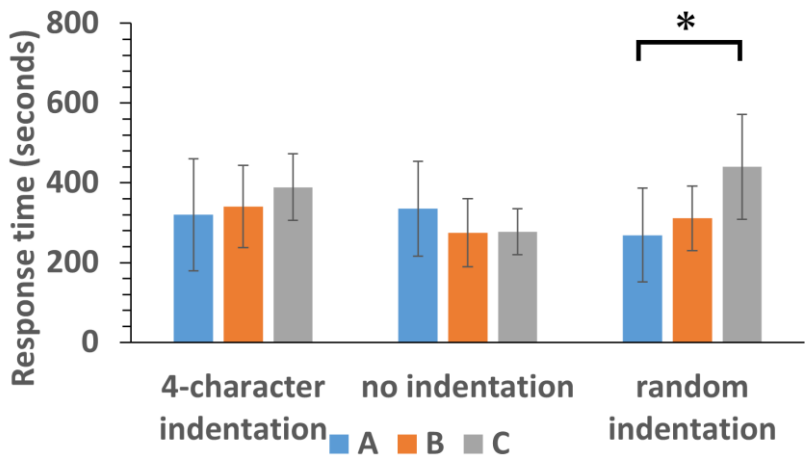

Figure 2: Response time

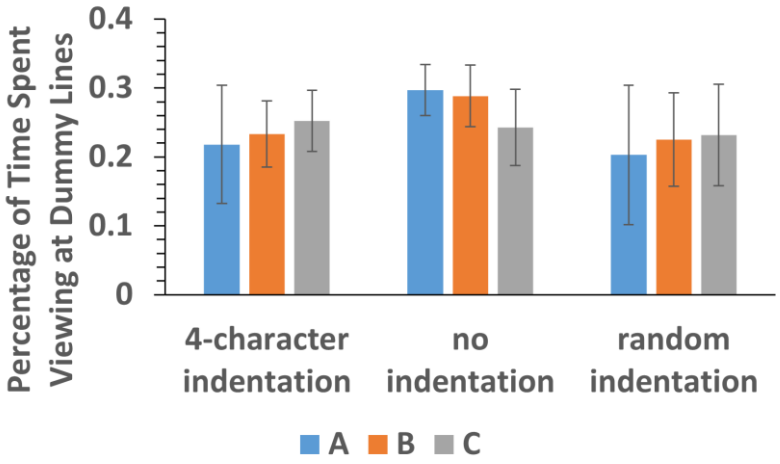

Figure 3: Percentage of time spent viewing "dummy lines"

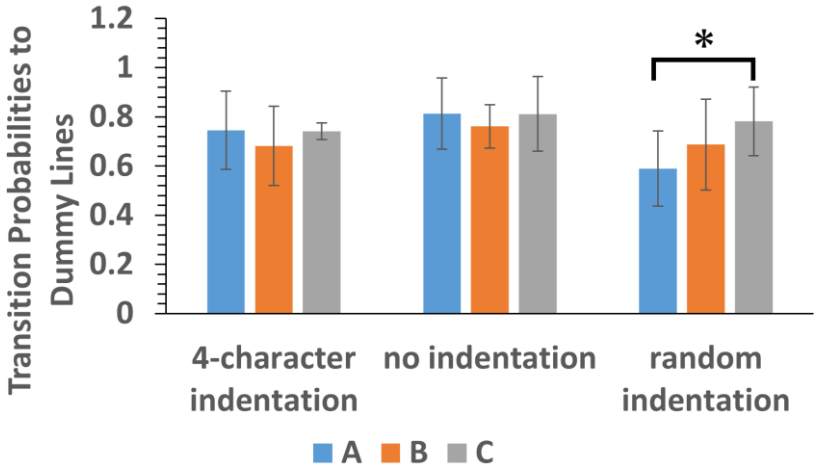

Figure 4: Transition probabilities to "dummy lines"

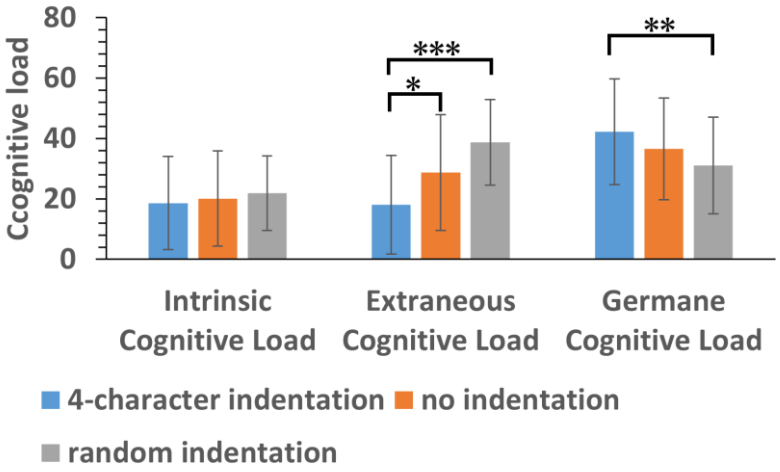

Figure 5: Cognitive load

5 Conclusion

The conclusion section provides a concise summary of the study, its objectives, methodology, and key findings. The authors restate the purpose of the study, which was to elucidate the impact of coding conventions on the readability of source code. They highlight the use of eye-tracking techniques to analyze the process of program comprehension and quantitatively demonstrate the influence of different types of coding conventions and their settings on reading efficiency.

The authors then summarize the key findings of the study, noting that through experimental procedures employing eye-tracking, the proportion of time spent viewing designated “dummy lines”, transition probabilities, and results from cognitive load questionnaires were examined. They report that the analysis suggested that the ambiguous term "readability," often associated with the use of indentation, could be more rigorously explained as considerations aimed at enhancing the accuracy of slicing for individuals less proficient in programming.

However, the authors also note that statistical differences were not observed with minor alterations such as the removal of indentation. They suggest that this finding indicates that to some extent, coding conventions may be disregarded in codes with small-scale and simplistic control structures, as utilized in the experiment.

Overall, the conclusion section effectively summarizes the key aspects of the study, including its objectives, methodology, and key findings. The authors successfully communicate the significance of their findings in elucidating the impact of coding conventions on the readability of source code and the potential implications for programming education and practice. The section is well-structured and concise, providing a clear and comprehensive overview of the study and its implications.

Future work will include examining differences arising from coding conventions other than indentation, as well as variations across programming languages.

Acknowledgement

This work was partly supported by Grant-in-Aid for Scientific Research (C) No. 22K02815 and No.23K02697 from the Japan Society for the Promotion of Science (JSPS).

References

[1] M. Carr, C. Wagner, “A study of reasoning processes in software maintenance Management,” Information Technology and Management, vol.3, pp.181–203, 2002.

[2] A. Goldberg, “Programmer as reader”. IEEE Software, 4(5),62, 1987.

[3] H. Kanamori, T. Higashimoto, Y. Yoneda and T. Akakura. Proposal of "Learning to Read Programs" in the Programming Process and Development of Learning Support System for the "Understanding Meaning" Process. IEICE Transactions on Information and Systems, 97(12), 1843-1846, 2014.

[4] T. Oshiro, Y. Matsuzawa and S. Sakai. (2011). Proposal of Code Reading Support Tool Using Tour Map. Proceedings of the 73rd National Convention, 441-442, 2011(1).

[5] M. Tanaka, T. Ishio and K. Inoue, Proposal of Additional Annotation Document Tag for Program Understanding. Technical Report of Software Engineering (SE), 2009(31(2009-SE-163)), 201-208, 2009.

[6] A. Hiratani, "Analysis Based on Control Dependence of Limited Program Reading Patterns Using Eye Movements," Bachelor's Thesis, Department of Intelligent Information Systems, Faculty of Information Engineering, Hiroshima Institute of Technology, 2020.

[7] R. Hanabusa, Y. Hayashi, M. Hirashima, and S. Matsumoto, “A Data Dependency-Based Analysis of Program Reading Patterns Using Eye Movements--For Programs Composed of Assignment and Arithmetic Operations. --,” Journal of Educational Systems and Information, Vol. 35, No. 2, pp. 192-203, 2018.

[8] R. Bednarik, M. Tukiainen, “An eye-tracking methodology for characterizing program comprehension processes” Proc. of the 2006 symposium on Eye tracking research & applications pp. 125–132, 2006