Automatic Mapping of Syntax Trees and Eye Movement for Semantic-based Program Comprehension Pattern Extraction

Haruhiko Yoshioka[*,a], Hidetake Uwano[b]

[a] dept. Advanced Information Engineering National Institute of Technology, Nara College Nara, Japan
[b] dept. Information Engineering National Institute of Technology, Nara College Nara, Japan

*ai1103@nara.kosen-ac.jp

Abstract

Understanding how efficient developers understand source code is an important study to improve the work and/or learning efficiency. Many studies have measured developers' eye movement to source code for understanding efficient reading methods. To understand the efficient reading method, researchers must comprehend the corresponding between eye movement and displayed source code. However, it is time-consuming to extract each understanding method/pattern among different source codes, because of the difference in control flow and formats. In this paper, the authors propose a method that converts the eye movement recorded as display coordinates to a semantic transition sequence based on a syntax tree.

Keywords: *programming education, program comprehension, eye tracking, pattern mining, knowledge extraction*

Introduction

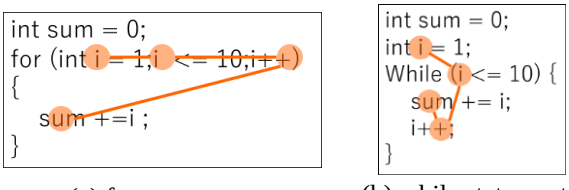
In software engineering research, understanding the "understanding process" during program reading is an important study to teach beginners the efficient reading methodology. The better understanding method improves the efficiency of development processes such as implementation and debug/testing and reduces development costs. A lot of previous research measures developers' eye movement to analyze the program reading (program comprehension) process (Hauser et al., 2020; Rodeghero et al., 2014; Crosby and Stelovsky, 1990; Bertram et al., 2020). In such research, eye movement is recorded as a time series of positional information by eye tracker and expressed as coordinates of the display. For this reason, some previous research analyzes developers' comprehension process based on their positional eye movement during source code reading (Hauser et al., 2020; Rodeghero et al., 2014; Crosby and Stelovsky, 1990; Bertram et al., 2020).

However, the position-based analysis of eye movement during program comprehension is a time-consuming analysis because researchers manually understand the correspondence between eye coordinates and source code. Analyzing more general understanding patterns is harder because different source codes have different control flows and formats. Table 1 shows an example of the data recorded by an eye tracker. X and Y mean two-dimensional coordinates that the participant "read" at each time. Previous studies used the position-based eye movement information and analyzed the developer's characteristics from the direction of eye movement and fixation time (the amount of time the gaze remains within a certain range). For example, Hauser et al. (2020) and Busjahn et al. (2015) found that skilled users tend to move their eyes up and down more than novices.

On the other hand, two source codes with the same control flow but different syntax, and eye movements are different even though the reviewer read the same control flow is the same (Figure 1). In the figure, circle and line mean the series of eye movements, each circle shows the fixation. Two code snippets in the figure calculate the summation from 1 to 10, the (a) uses a for statement and the (b) uses a while statement. These two statements have different structures, hence coordinate-based analysis appears the eye movements for two source codes differently, such as "left to right" at for statement and "up to down" at while statement. Previous studies manually

Table1. Eye movements as display coordinates

Time	X	Y
24:54.1	322	457
24:54.1	322	458
24:54.2	328	463
24:54.2	326	469
24:54.2	320	479



(a) for statement (b) while statement

Fig1. Two code snippets with different statement

interpret these different eye movements to come from the same understanding pattern, the interpretation needs a lot of time to find.

In this paper, the authors propose an analysis method that converts coordinate-based eye movement to transitions for syntax node in source code. The proposed method abstracts differences in coordinates and statements by representing eye movement as transitions between semantic units in source code. Figure 1 shows two different eye movements onto two source codes, both source codes calculate the summation from 1 to 10. In the figure, eye movement for (a) and (b) follows the same processes, initialization, condition, increment, and result calculation. The simple coordinate-based analysis describes these eye movements as different movements. Our proposed method extracts an ordered list of syntax nodes that eye movement passed. The method allows us to use pattern mining techniques for extracting a general understanding strategy from eye movement data. In this paper, the authors use syntax nodes to represent the smallest unit of processing content (token) that eye movement passed. More abstract representation of eye movement is an important future work for efficient large-scale analysis.

Related Works

Eye-tracking is well used to analyze differences between novice and expert programmers. The state in which the gaze remains within a certain range for a certain time is called fixation, and its central coordinates are called the fixation point. Fixation points are used to identify the location of the reader's interest and its change over time. Hauser et al. compared the differences in the gaze-to-stop point between novices and experts. Hauser et al. (2020). The results showed that the expert group tended to read source code in a nonlinear manner while the novice group read linearly.

Area of interest (AOI) is another well-used analysis method. The researcher defines multiple AOIs on the object (picture or source code snippet), then successive fixations within the same AOI are summarized. Several program comprehension research analyzes developers' comprehension patterns by defining AOIs on the words and phrases in source code (Rodeghero et al., 2014; Crosby and Stelovsky, 1990; Bertram et al., 2020; Chandrika et al., 2017). Rodeghero et al. (2014) analyzed the length of fixation time for AOIs assigned to method signatures, method calls, control flow, and others. The analysis showed that programmers read method signatures much longer time than (method calls) and control flow. Crosby et al. analyzed the difference in review time allocation for each AOIs between novices and experts. The result showed that the experts spent less time on comments than novices (Crosby and Stelovsky, 1990). Peitek et al. (2020) converted coordinate-wise eye tracking data to line-wise for comparison between novice and intermediate students. The results showed that intermediate students moved their gaze to the top line of the source code more frequently than novice students.

Most previous research converted coordinate-based eye tracking data to AOI-based or line-based by manually determining which area in the source code is the AOI/line of code. This conversion requires a lot of time for longer source code. One of the open-source software, iTrace automatically converts the coordinate-based eye tracking data into the line/column numbers in source code. iTrace is implemented as a plugin of Eclipse, one of the commonly used in development and education. The authors use iTrace to convert coordinate-based eye tracking data to line/column numbers of source code. Our proposed method extracts words/characters of the source code from the line/column numbers, then make corresponds with syntax tree.

Proposed Method

Our proposed method outputs eye movement as a transition of syntax node in the source code. Figure 2 shows an overview of our method. In the figure, each square represents software and hardware module, each thin arrow represents information flow. The developer's eye movement during the code reading is measured using an **Eye Tracker**. The Eye Tracker outputs eye movement as a time-series display coordinate (such as X:121, Y:313) with observed time. **Coordinate Line/Column Converter** receives the coordinate-based eye movement and the source code as inputs, then outputs the line/column-based eye movement (Main.java, L:1, Row:13). We used iTrace as the Coordinate Line/Column Converter. Source code is also sent to **Java Parser** to extract Syntax Tree from the source code. Our implementation used ANTLR Java parser, an open-source parser generator. **Syntax Tree/Eye Linker** receives the syntax tree of the source code and line/column-based eye movement, and outputs syntax-node-based eye movement. Java Parser performs lexical and syntactic analysis, and Syntax Tree/Eye Linker converts eye movements to word units in the source code by mapping lexical analysis results to line/column-based eye movements. Furthermore, eye movement is expressed in node units on the syntax tree by mapping the parse results.

Figure 3 shows an example of the syntax tree, that is generated from the main method of source code in Figure 4. In Figure 3, each leaf node denotes words in the source code and its Line/Column numbers. Each internal node describes a syntax structure of child nodes. For example, the right most internal node (block @19) describes that formed from three words (sum, +=, i) at Line 5 in Figure 4, and is a part of for statement (statement @13) at Line 4. The proposed method systematically converts each fixation into the meaning of a word in the source code, the output can use for different analyses with minimal post-processing.

Case Study

In this section, we describe how the proposed method can be used to analyze the developer's eye movement. We select Main.java in Figure 4 and eye movement for

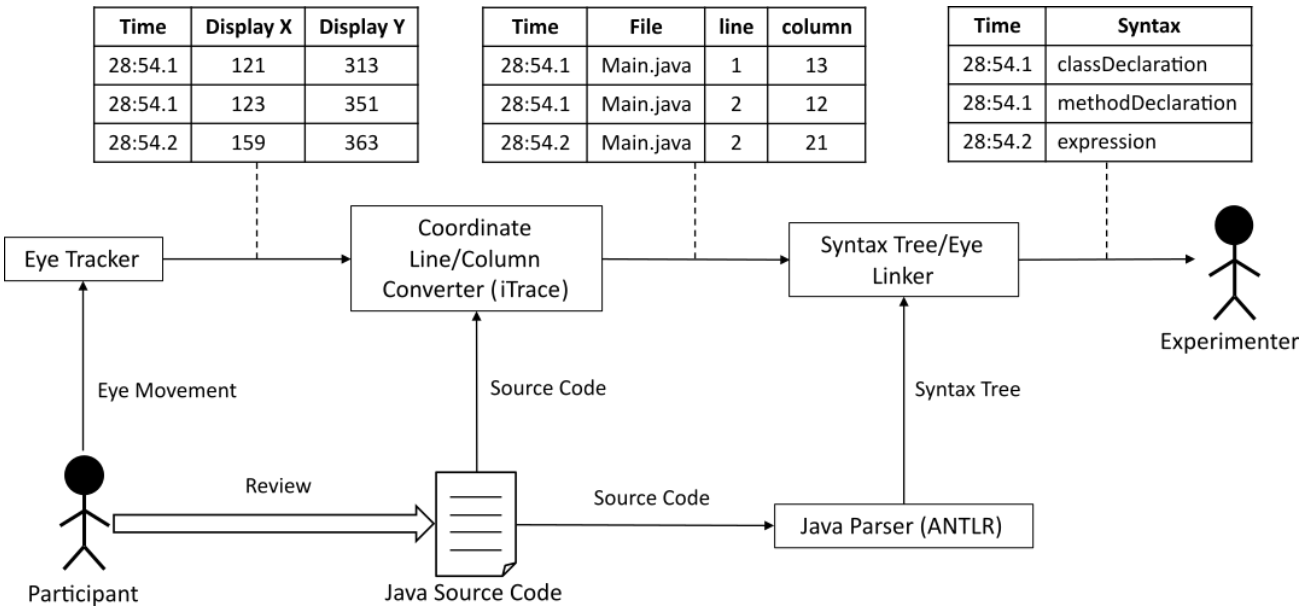


Fig2. Proposed Method

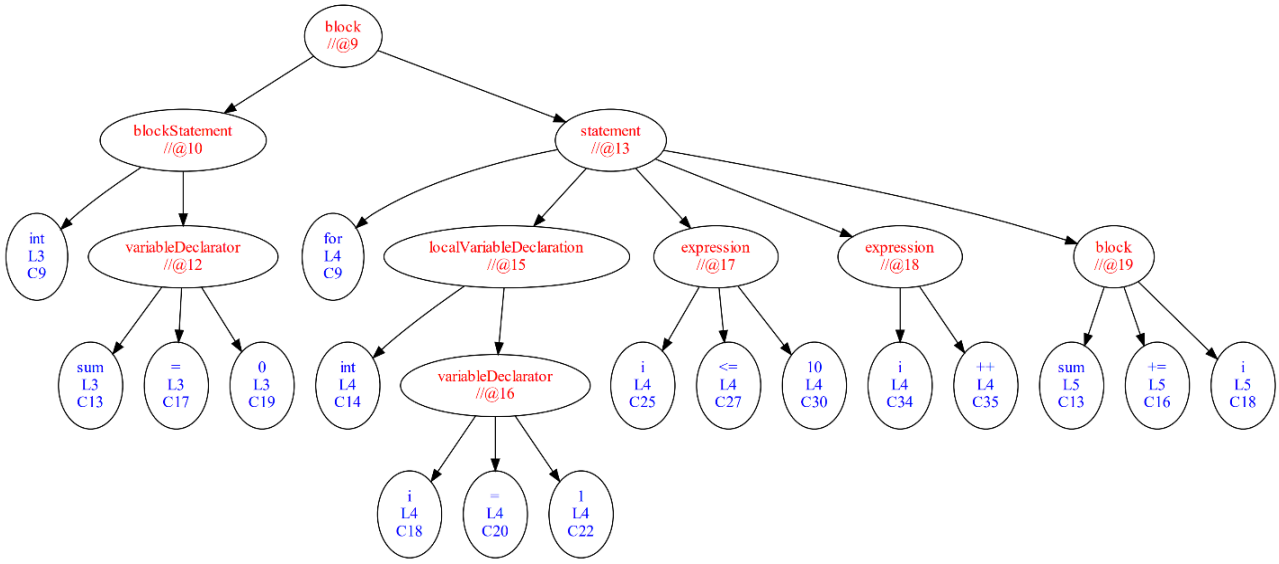


Fig3. An Example of Syntax Tree

the source code for our case study. Figure 5 shows the eye movement visualization by our implementation system, and Table 2 shows an output of the implementation of the proposed method. **Syntax Tree** column shows text form expression of syntax tree for each token. Visualized coordinate-based eye movement in Figure 5 clearly shows the participant read the index declaration, conditional expression, index incrementation, and line 5 in this order. That is the participants may try to understand a process in the "for" statement or a role of variable i. Line/column number and token in Table 2 describe the same eye movement, however it is hard to understand what syntax structure is

```
01 public class Main {
02     public static void main(String[] args) {
03         int sum = 0;
04         for (int i = 1; i <= 10; i++) {
05             sum+=i;
06         }
07         System.out.println(sum);
08     }
09 }
```

Fig4. Main.java

read by the participant without the source code. Compared to these previous eye movement descriptions, the text form syntax tree includes the structure information with different granularities. Hence the proposed method allows for researchers following analysis:

- **Summarization by syntax structure**
  Every syntax tree text in Table 2 includes "blc@9/ stm@13", which means the entire eye movement concentrated on the "for" block in lines 4-6. Each number describes a unique identifier of syntax structure such as class, method, block, and statement, hence consecutive eye movements that have the same ID can combine as one long-time fixation.

```
1 public class Main {
2     public static void main(String[] args) {
3         int sum = 0;
4         for (int i = 1; i <= 10; i++) {
5             sum+=i;
6         }
7         System.out.println(sum);
8     }
9 }
```

Fig5. Eye Movement for Main.java

- **Abstracted sequential analysis**
  Figure 6 shows the same eye movement in Figure 5, however, is written at the syntax tree in Figure 3. A thin dotted line shows an order of tokens in which eye movements were fixated. Researchers can find the order from line/column-based eye movement, in this case, we can find the eye movement follows variable i in the "for" statement. A thick dotted line in Figure 6 shows an abstracted eye movement. The eye movement follows index declaration (VariableDeclaration@16), conditional expression (conditional expression@17), index increment (expression@18), and the statement in for block (block@19), in that order. Table 2 also shows the syntax tree text includes these IDs and tokens; researchers can select abstraction level that research purpose requires.

- **Pattern mining of vectorized syntax structure**
  The syntax tree text includes different abstraction levels of token information that eye movements stayed. Our proposed method outputs the text

Table2. An Output Example of the Proposed Method

#	Time	Fixation	Line	Column	Token	Syntax Tree
1	02:15	705, 226	4	18	i	blc@9/ stm@13/ lvdec@15/ vdec@16/ i[L4,C18]
2	02:16	800, 235	4	22	1	blc@9/ stm@13/ lvdec@15/ vdec@16/ 1[L4,C22]
3	02:18	877, 234	4	25	i	blc@9/ stm@13/ exp@17/ i[L4,C25]
4	02:19	928, 228	4	27	<=	blc@9/ stm@13/ exp@17/ <=[L4,C27]
5	02:21	1076, 237	4	34	i	blc@9/ stm@13/ exp@18/ i[L4,C34]
6	02:23	708, 283	5	18	i	blc@9/ stm@13/ blc@19/ i[L5,C18]
7	02:24	618, 282	5	13	sum	blc@9/ stm@13/ blc@19/ sum[L5,C13]

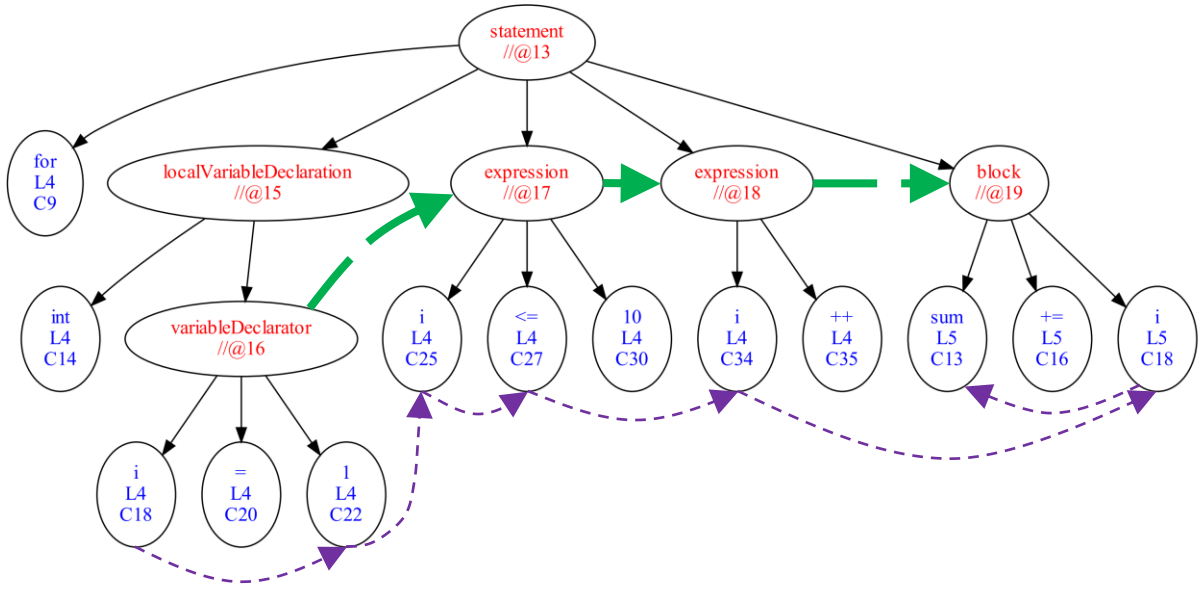


Fig6. An Eye Movement Visualization with Syntax Tree

pautomatically without any pre-analysis or preparation such as AOI definitions. Hence the pattern mining analysis of vectorized eye movement information from syntax trees will provide important data to analyze their reading pattern or method.

- **Inter-source code comparison and mining**
  Different source codes may use different identifiers for the same purpose, such as "sum" and "total" for a variable that contains a total number of calculations. Text-based pattern mining cannot find patterns that contain different identifiers with the same purpose/usage. Also, coordinate-based eye movement cannot find such patterns because the positional relationship of tokens is different between source codes, even if the two identifiers are used for the same purpose. On the other hand, abstracted, and vectorized syntax structures from syntax tree text can extract such patterns.

Conclusion

In this paper, the authors proposed an analysis method that converts coordinate-based eye movement to transitions for syntax nodes in source code. The proposed method abstracts differences in coordinates and statements by representing eye movement as transitions between nodes in a syntax tree. Our case study showed that the text form syntax tree is useful to summarize consecutive eye movements and abstract the eye movement position from token to a statement or a block.

Future work of this study includes the following analyses using the proposed method.

- Extract common understanding patterns from developers' eye movement
- Efficiency and effect analysis of eye movement patterns including comparison between experts and novices
- Pattern mining of eye movements to different source codes

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP21K11842.

References

Bertram, I., Hong, J., Huang, Y., Weimer, W., and Sharafi, Z. (2020). Trustworthiness perceptions in code review: An eye-tracking study. In Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 1-6.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., and Tamm, S. (2015). Eye movements in code reading: Relaxing the linear order. In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC), 255–265.

Chandrika, K. R., Amudha, J., and Sudarsan, S. D. (2017). Recognizing eye tracking traits for source code review. In Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 1–8.

Crosby, M. and Stelovsky, J. (1990). How do we read algorithms? a case study. Computer, 23(1), 25–35.

Guarnera, D. T., Bryant, C. A., Mishra, A., Maletic, J. I., and Sharif, B. (2018). iTrace: Eye tracking infrastructure for development environments. In Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications (ETRA), 1-3.

Hauser, F., Schreistter, S., Reuter, R., Mottok, J. H., Gruber, H., Holmqvist, K., and Schorr, N. (2020). Code reviews in C++: Preliminary results from an eye tracking study. In Proceedings of the ACM Symposium on Eye Tracking Research and Applications (ETRA), 1-5.

Peitek, N., Siegmund, J., and Apel, S. (2020). What drives the reading order of programmers? An eye tracking study, In Proceedings of the 28th International Conference on Program Comprehension (ICPC), 342–353.

Rodeghero, P., McMillan, C., McBurney, P. W., Bosch, N., and D'Mello, S. (2014). Improving automated source code summarization via an eye-tracking study of programmers. In Proceedings of the 36th International Conference on Software Engineering (ICSE), 390–401.