Analysing program source code reading skills with eye tracking technology

Vilius Turenko, Simonas Baltulionis, Mindaugas Vasiljevas, Robertas Damaševičius
Department of Software Engineering
Kaunas University of Technology, Kaunas, Lithuania
robertas.damasevicius@ktu.lt

***Abstract*—Many areas of software engineering require good program code reading skills. We analyse the process of program reading using gaze tracking technology. We performed a study with six subjects, who performed four code reading tasks. The errors the embedded into program sources code and the lines of code with the areas were analysed as Areas of Interest (AoI). We formulated a research hypothesis and tested it using a one-way analysis of variance (ANOVA) test. The results of the study confirmed our research hypothesis that the number of fixations on AoI is larger than the number of fixations on other areas.**

Keywords—program comprehension, code reading; eye tracking, gaze tracking, human-centered computing.

I. Introduction

Program code reading skills are important in many areas of software engineering, especially, in adopting good code writing practices and techniques, understanding how programs work, identifying cases of poor programming style and bad design, and delivering effective software maintenance. Examples include program tracing and searching for bugs, code smells and design anti-patterns [1]. As automatic methods for finding bugs and poor coding practices are still not very effective [2], source code reading and analysis by human experts remain as relevant as ever. Program comprehension is a crucial part of computer science education, providing an important part of an understanding of complexity of information technology (IT) systems [3]. The interest on applying gaze tracking in the context of multimedia supported learning is on the rise [4]. Gaze data had been successfully applied to analyze changes in cognitive load during assimilation of learning materials and are starting to be incorporated into adaptive e-Learning systems [5]. However, currently there are no effective strategies in evaluating code reading skills and assessing program comprehension. Recently, eye tracking and was proposed as a viable research instrument for evaluating source code reading [6]. The outcomes of gaze tracking studies are especially relevant in the context of Evidence-based Software Engineering (EBSE) in order to provide detailed insights regarding different practices in software engineering [7].

Eye movements are directly related to cognitive and information processing processes, and through these processes, visual information is used to stimulate the brain and to understand the given task. There are two assumptions related to cognitive processes and fixations: 1) if a person is seeing an object (such as a word), he/she tries to understand it; 2) a person fixates his/her gaze on an object until he/she understands it. A fixation is an aggregation of gaze points based on a specified area and time span. An Area of Interest (AoI) is a part of a visual stimulus that is of special importance Other important characteristics are a scan path, which is a series of fixations that indicate the path and tendency of eye movements, and a heat map, which identifies the focus of visual attention [8]. For example, Uwano et al. [9] studied graduate students conducting code reviews and discovered that their gaze patterns followed a common scanpath, first reading code top to bottom, and then rereading a few parts in more depth. Chandrika et al. [10] confirmed the positive relationship of eye tracking traits over source code lines and comments for code comprehension. Melo et al. [11] analysed how programmers debug code with embedded pre-processor directives. Jbara and Feitelson [12] analysed how code repeatability impacts the number of fixations in a predefined area of interest (AOI), and the total fixation time. Beelder and Plesis [13] analysed how the number and durations of fixations are influenced by syntax highlighting. Yennigall et al. [14] also used fixation counts and duration to analyse how programming novices understood program code.

In this paper, we describe the results of gaze tracking study on evaluating and analysing the code reading skills of software programmers, specifically focusing on the ability to find errors in program code.

II. Methodology

A. Program reading tasks

The study consisted of 4 tasks:

a. In Task 1, the aim was to read the program source code with the aim of finding the result returned (printed) (Fig. 1).

b. In Task 2, the aim was to identify the purpose of the algorithm and discover the hidden error associated with the incompatibility of the variable types (Fig. 2).

c. In Task 3, the aim was to find three syntactic errors related to the incorrect use of variable names, types and basic methods (Fig. 3).

d. In Task 4, the aim was to determine whether the algorithm would perform the specified function, and to find a hidden semantical error (Fig. 4).

```

/*Duota klasė stačiakampis, kuriojeduoti ilgio ir ploč
 io parametrai,
*klasėje užklogi operatoriai daugybos ir dalybos vei
 ksmams
*
*/
public class Stačiakampis
{   //Kintamieji
    public int ilgis { get; set; }
    public int plotis { get; set; }

    //Konstruktorius
    public Stačiakampis(int ilgis, int plotis)
    {
        this.plotis= ilgis;
        this.ilgis  =plotis ;
    }

    //Ilgių daugyba
    public static int operator *(Stačiakampis x, Sta
   čiakampis y)
    {
        return x.ilgis / y.ilgis;
    }
    //Pločių dalyba

    public static int operator /(Stačiakampis x, Sta
    čiakampis y)
    {
        return x.plotis * y.plotis;
    }

}
class Program
{
    static void Main(string[] args)
    {
        Stačiakampis s1 = new Stačiakampis(1, 5);
        Stačiakampis s2 = new Stačiakampis(5,10);
        Console.WriteLine("{0} {1}", s1 / s2, s2 * s1
   ); }}

                ATS 5 0
```

Fig. 1. Program source code with Area of Interest (AoI) highlighted for Task 1: calculate output of a program

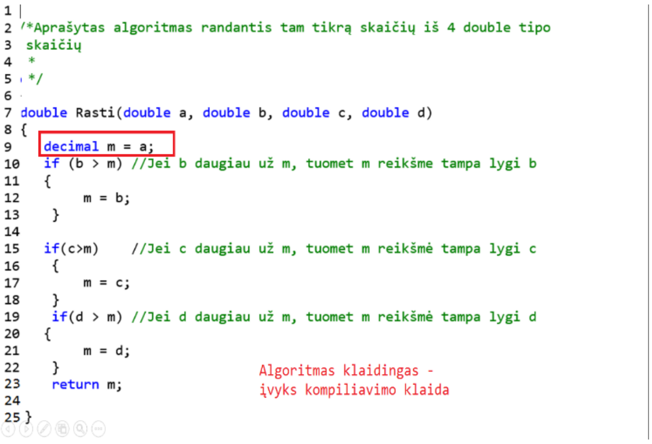

Fig. 2. Program source code with Area of Interest (AoI) highlighted for Task 2: find syntactic error

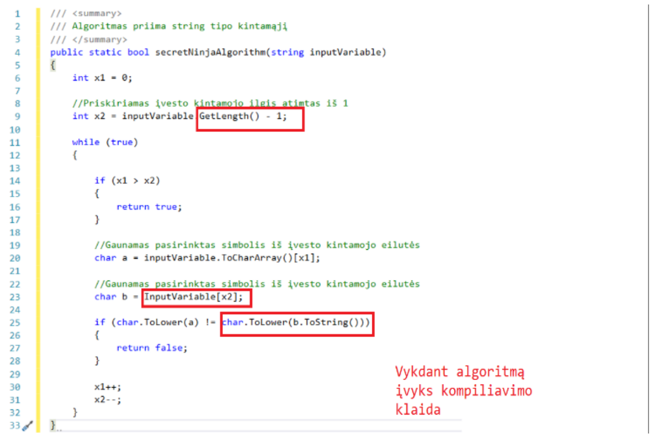

Fig. 3. Program source code with Area of Interest (AoI) highlighted for Task 3: find multiple syntactic errors

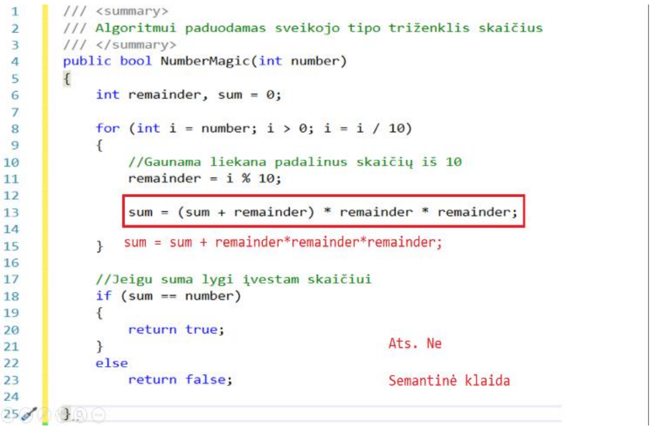

Fig. 4. Program source code with Area of Interest (AoI) highlighted for Task 4: find semantic error

B. *Data collected by gaze tracking*

During gaze tracking we collect the number and location of fixations, which are gaze points that are directed towards a certain part of an image, which is labelled as Area of Interest (AoI). Fixations are indications of visual attention. Here we analyze the distribution of the number of fixations between and out of AoIs. The eye movements between fixations are known as saccades. However, we do not use the saccade data in this study. A scan path is a directed path created by saccades between eye fixations.

C. *Research hypotheses*

We assume that subjects are thinking about the object of interest when they are looking directly at it. Based on this assumption, we formulate the following research hypothesis:

H1: The number of fixations on Areas of Interest is larger than the number of fixations on other areas.

D. *Testing of hypotheses*

For testing of hypotheses we employ a statistical one-way analysis of variance (ANOVA) test. The test, which is a standard statistical method, confirms or rejects the equality of the averages of two or more samples by examining the variances of samples. ANOVA compares the variance between the data samples to variance within each particular sample. If the between-sample variance is much larger than the within-sample variance variation, the average values of different samples can not be equal.

III. Experimental Setting and Results

A. *Experimental settings*

Six participants (1 female and 5 male) were recruited for this study, ages between 20 and 25 with an average of 22.8 years. All participants had normal or corrected-to-normal vision. Participants were familiar with computers and had previous experience in using the internet and all of them were studying or working in programming sphere. An informed consent was obtained from subjects before the study.

All subjects were given the same laptop Dell which had an additional monitor used for experiment and the Tobii Eye Tracker 4C eye-tracking device used to record eye movements and gaze fixations. The eye tracker uses infrared corneal reflection to measure point of gaze with data rates of 90 Hz. A 24 inch screen was used to show the slides which consisted of programming source code. The eye tracker using instructions was mounted just below the visible screen area. The operating distance between the eye tracker and subjects' eyes was between 70-75 cm. Efforts were made to ensure good lighting and a device calibrated before the test. For each subject the eye tracker was re-calibrated using an integrated 5-point calibration to achieve most accurate results.

Before the start of the experiment, the subjects were asked to fill in the Google Form questionnaire before the start of the study on their demographical characteristics (gender, education, age, level of programming skills). All responses were anonymized. After filling personal characteristics subjects had a chance to read some common information about tasks that they will face in this experiment, this way subjects were informed about some important rules, for example, no additional libraries or other extensions were used, also that some tasks were bug free and some had hidden bugs, the idea was to stimulate the subjects to be focused by not telling what tasks had bugs and what were bug free. After introducing tasks in common, the presentation with the slides containing the source code of tasks was opened, the observation session started at the start of each task and the session was stopped after the task was completed, each task had a separate observation session. 3 and 4 tasks had some brief information about given algorithms, for example, definition of palindrome and Armstrong's number and examples of each case. To complete each task, 90 seconds were given. After the completion of each task, the participants were asked to provide the answers in a Google Form on what is the result of program execution (Task1), what is the purpose of an algorithm (Task 2), and is the program correct (Task 3, Task 4).

B. Experimental system

Gaze monitoring system was used to measure the number and duration of fixations in the Areas of Interest (AOIs). The system consists of components listed below (see Fig. 5).

- The Data Gathering Module reads the raw gaze data from the eye tracker device via USB.
- The Data Preprocessing Module filters noise, calculates additional metrics and characteristics like saccades.
- The Data Persistence Module saves the acquired gaze data to CSV, XML or database.
- The Data Post-processing Module maps persisted gaze data to AOIs and calculates additional data features such as the total and average number and duration of fixations.
- The Configuration Module configures how data is gathered and persisted in the system.

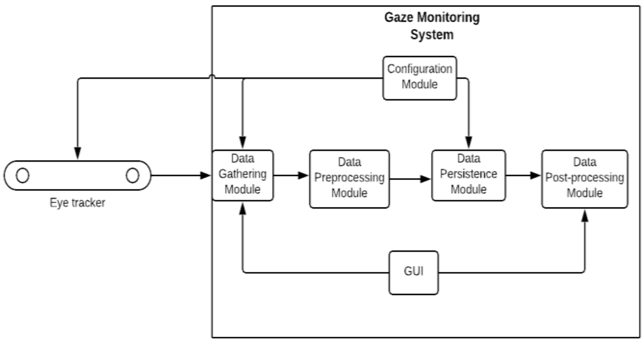

Fig. 5. Architecture of a system

System offers four types of data stream which are used to gather fixations and saccades directly from gaze tracking device.

- Unfiltered gaze
- Lightly filtered gaze
- Sensitive fixation
- Slow fixation

For this experiment, sensitive fixation type was chosen because of its accuracy and unnecessary noise reduction.

In addition, the system is running in the background and it has no effect on the stimulus, thus the subject's attention is concentrated only to source code.

Besides types of data stream, before starting gaze tracking session, user has an option to choose to record his screen, but for now it is only a prototype version, which needs to be improved for better accuracy, also session can have additional information about subject, for example name, age and other description, if it is not necessary, user can select anonymous session. In the near future, system will offer an option to choose screen resolution manually, which will allow to select concrete zones of interest.

C. Results

The results of participants (number of fixations) are summarized according to tasks and subjects in Fig. 6.

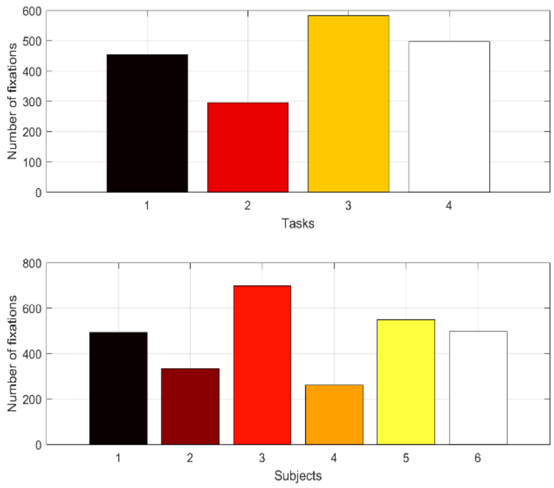

Fig. 6. Summary of the number of fixations according to subjects and tasks

An example of the gaze path generated from gaze tracking data is presented in Fig. 7. The gaze path shows how and in what sequence the subject has read the code. Note the order of reading is clearly not linear.

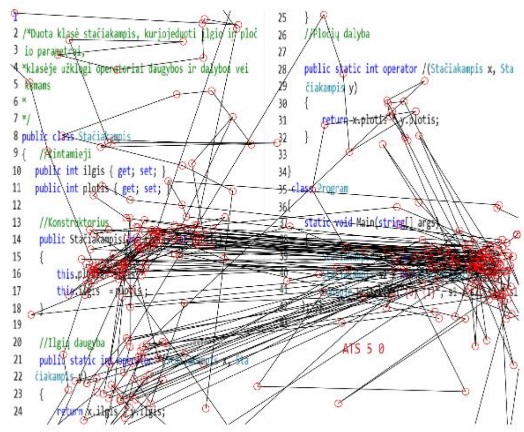

Fig. 7. Example of a gaze path (Task 1, Subject 1)

An example of the heatmap generated from gaze tracking data is presented in Fig. 8. Note that most of attention was focused on and around the Area of Interest centred on code line 42 (see also Fig. 1).

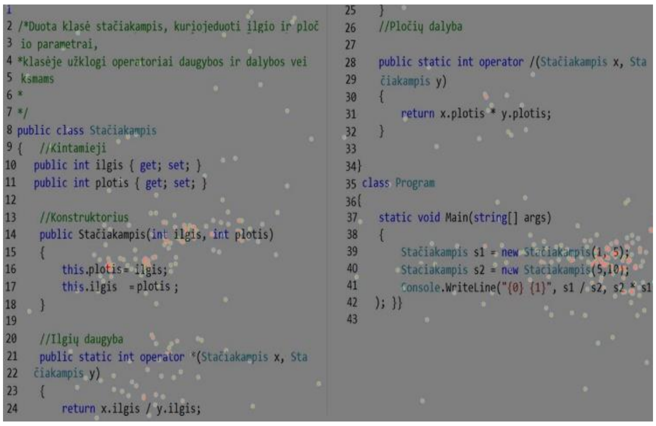

Fig. 8. Example of a gaze fixation heatmap (Task 1, Subject 1)

In Fig. 9, the average gaze fixation numbers for AoI and Non-AoI areas is presented. We can see that for all tasks, the number of fixations on AoIs was larger, although the difference was not statistically significant for Task 2 (also see the results of statistical testing using ANOVA in Table I).

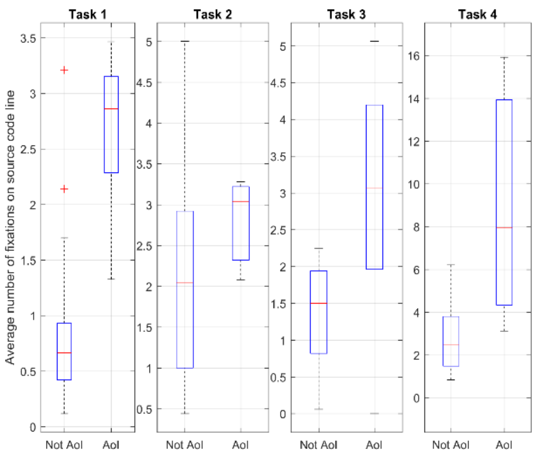

Fig. 9. Average number of fixations on AoI vs non-AoI source code lines

The results of statistical testing using ANOVA are presented in Table I. We found statistically significant differences in the number of fixations on the Areas of Interest (AoI) vs non-AoI for Tasks 1, 3 and 4. However, we did not find such differences for Task 2.

TABLE I. RESULTS OF STATISTICAL TESTING

Task	**Results of ANOVA**	
	F-value	***p-value***[a]
1	37.79	0 (***)
2	0.66	0.4245
3	14.73	0.0006 (***)
4	15.58	0.0006 (***)

a. *** - statistically significant

D. *Limitations and threats to validity*

The study is based on the assumption that humans think about objects when look at them, however we cannot be sure that is assumption is correct. Our eye-tracking experiment only explores the processing of cognitive response to visual stimulus without considering the quality of responses. Moreover, due to a small sample of subjects and gender bias (all participants were male) we could not analyse the gender and affective differences, which have been noted as significant in other gaze tracking studies [15]. To minimize threats to validity, the participants did not know about the hypothesis formulated for the research. They only knew that they would be in helping us to understand how program code is read and understood.

In three tasks of four performed we were able to confirm our research hypothesis. In one, task the hypothesis could not be confirmed. We think that we reason was in poor design of the task, which we hope to improve in our further research.

IV. CONCLUSION

We have presented a study aimed at comprehending how programmers read and debug program code. Our results indicate that gaze tracking can be used successfully to follow and assess the cognitive behaviour of programmers as they correctly identify the errors embedded into the source code. The number of gaze fixations is a significant parameter when assessing the level of attention attributed to a particular Area of Interest.

Future work will focus on the methodological improvement of our research study and collection of a larger dataset of data from more subjects.

REFERENCES

[1] Obaidellah, U., Al Haek, M., & Cheng, P. C. (2018). A survey on the usage of eye-tracking in computer programming. ACM Computing Surveys, 51(1) doi:10.1145/3145904

[2] Gupta, A., Suri, B., Kumar, V., Misra, S., Blažauskas, T., & Damaševičius, R. (2018). Software Code Smell Prediction Model Using Shannon, Rényi and Tsallis Entropies. Entropy, 20(5), 372. doi:10.3390/e20050372

[3] Damaševičius, R. (2009). On The Human, Organizational, and Technical Aspects of Software Development and Analysis. In Information Systems Development (pp. 11–19). Springer US. doi:10.1007/b137171_2

[4] Alemdag, E., & Cagiltay, K. (2018). A systematic review of eye tracking research on multimedia learning. Computers and Education, 125, 413-428. doi:10.1016/j.compedu.2018.06.023

[5] Rosch, J. L., & Vogel-Walcutt, J. J. (2013). A review of eye-tracking applications as tools for training. Cognition, Technology and Work, 15(3), 313-327. doi:10.1007/s10111-012-0234-7

[6] Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. In Proceedings of the 11th Koli Calling International Conference on Computing Education Research - Koli Calling '11. ACM Press. doi:10.1145/2094131.2094133

[7] Sharafi, Z., Soh, Z., & Guéhéneuc, Y. (2015). A systematic literature review on the usage of eye-tracking in software engineering. Information and Software Technology, 67, 79-107. doi:10.1016/j.infsof.2015.06.008

[8] Blascheck, T., Kurzhals, K., Raschke, M., Burch, M., Weiskopf, D., & Ertl, T. (2017). Visualization of eye tracking data: A taxonomy and survey. Computer Graphics Forum, 36(8), 260-284. doi:10.1111/cgf.13079

[9] Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In Proceedings of the 2006 symposium on Eye tracking research & applications - ETRA '06. ACM Press. doi:10.1145/1117309.1117357

[10] Chandrika, K. R., Amudha, J., & Sudarsan, S. D. (2017). Recognizing eye tracking traits for source code review. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE. doi:10.1109/etfa.2017.8247637

[11] Melo, J., Narcizo, F. B., Hansen, D. W., Brabrand, C., & Wasowski, A. (2017). Variability through the Eyes of the Programmer. In 2017

IEEE/ACM 25th International Conference on Program Comprehension (ICPC). IEEE. https://doi.org/10.1109/icpc.2017.34

[12] Ahmad Jbara and Dror G. Feitelson. 2015. How programmers read regular code: a controlled experiment using eye tracking. In Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension (ICPC '15). IEEE Press, Piscataway, NJ, USA, 244-254.

[13] Beelders, T., & du Plessis, J.-P. (2016). The Influence of Syntax Highlighting on Scanning and Reading Behaviour for Source Code. In Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists on - SAICSIT '16. ACM Press. https://doi.org/10.1145/2987491.2987536

[14] Yenigalla, L., Sinha, V., Sharif, B., & Crosby, M. (2016). How Novices Read Source Code in Introductory Courses on Programming: An Eye-Tracking Experiment. In Lecture Notes in Computer Science (pp. 120–131). Springer International Publishing. https://doi.org/10.1007/978-3-319-39952-2_13

[15] Ksiazeh, K., Marszalek, Z., Capizzi, G., Napoli, C., Polap, D., & Wozniak, M. (2019). Faster image filtering via parallel programming. International Journal of Computer Science & Applications, 16(1), pp. 55-67.

[16] Liaudanskaitė, G., Saulytė, G., Jakutavičius, J., Vaičiukynaitė, E., Zailskaitė-Jakštė, L., & Damaševičius, R. (2019). Analysis of affective and gender factors in image comprehension of visual advertisement. Artificial Intelligence and Algorithms in Intelligent Systems. CSOC2018. Advances in Intelligent Systems and Computing, vol 764. Springer, Cham, 1-11. doi:10.1007/978-3-319-91189-2_1