Attention Patterns for Code Animations: Using Eye Trackers to Evaluate Dynamic Code Presentation Techniques

Louis Spinelli*
Information School
University of Washington
Seattle, WA, USA
spinelli@uw.edu

Maulishree Pandey*
School of Information
University of Michigan
Ann Arbor, MI, USA
maupande@umich.edu

Steve Oney
School of Information
University of Michigan
Ann Arbor, MI, USA
soney@umich.edu

ABSTRACT

Programming instructors seek new ways to present code to novice programmers. It is important to understand how these new presentation methods affect students. We prototyped three different ways to animate the presentation of code. We used eye-tracking technology to observe participants as they were presented with animations and completed three activities: code summarization, syntax error correction, and logic error correction. The prototypes, our method for observation, and our analysis methods were each informed by previous research. We observed variation in how participants consumed animations. Our initial results indicate that viewing animations of a single textual representation of source code may affect the attentional processes of novice programmers during subsequent tasks.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**; Distance learning; • **Human-centered computing** → *Visualization design and evaluation methods*;

KEYWORDS

Program Animation, Programming Education, Eye Tracking

ACM Reference Format:
Louis Spinelli, Maulishree Pandey, and Steve Oney. 2018. Attention Patterns for Code Animations: Using Eye Trackers to Evaluate Dynamic Code Presentation Techniques. In *Proceedings of 2nd International Conference on the Art, Science, and Engineering of Programming (<Programming'18> Companion).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3191697.3214338

1 INTRODUCTION

Since the 1940s, visual representations of program code have evolved from static flow charts to dynamic animations[20]. A large body of research is now focused on understanding program visualization in introductory programming education[19]. The belief that students experience difficulty learning because they lack a concrete mental model has often been cited as a reason for developing program visualizations[18]. Two additional reasons given in support of dynamic animations are the ease of describing program behavior and removing the potential for an instructor to make mistakes when presenting code[1].

Our study provides insights into the effects of animating the presentation of textual source code and directions for future research. In this study, animating the presentation of textual source code ("a single textual representation") refers to animating the text of the source code without the addition of separate graphical representations of elements contained within the code such as variables, classes, and methods.

Animations of source code may have an effect on the participants' reading approach and attentional processes. Element coverage, which is the fraction of words a participant fixates on when reading, is one part of a novice programmer's reading approach [8]. Previous work has found element coverage for expert programmers is lower than novice programmers because expert programmers are able to focus on the relevant words within the code [8]. For novice programmers, watching an animation that illustrates the non-linear nature of code may reduce element coverage during later tasks. A participant's ambient and focal vision correspond to bottom-up and top-down attentional processes, respectively, and are measured using the K-coefficent [14]. Ambient vision, indicated by negative K-coefficient values, is beneficial for exploring code whereas focal vision, indicated by positive K-coefficient values, is beneficial for the inspection of elements within code [14].

In order to determine how code animations might affect these attention patterns, we recorded and examined the eye-gaze of novice programmers for three different animation interventions. We observed that our participants consume animations differently, especially when they were given control of the animation speed. For all animation interventions where text was legible, we observed an increase in element coverage per minute for all participants as they viewed the animation. However, this increase does not appear to carry over to programming activities after the animation has finished. We observed a reduction in the element coverage during debugging tasks for participants who watched non-linear animations. Watching an animation also appears to have an effect on participants' K-Coefficient. We observed an increase in focal attention during debugging and summary tasks and an increase in ambient attention during logic tasks.

*The first and second authors contributed equally to this project

<Programming'18> Companion, April 9–12, 2018, Nice, France

ACM ISBN 978-1-4503-5513-1/18/04…$15.00
https://doi.org/10.1145/3191697.3214338

2 NOVICE PROGRAMMERS AND THE PRESENTATION OF SOURCE CODE

2.1 Background

A large body of research is focused on understanding program visualization in introductory programming education [16, 19]. Another body of literature combines eye-tracking and programming education research [2, 3, 7, 22]. Existing eye tracking studies have explored how novice programmers interact with single textual representations of static source code [8, 10, 12, 23], multiple representations of code, [5, 6], and the effect of animation speed on comprehension [13].

Previous research has focused on helping novice programmers understand run-time dynamics of computer programs through visualizations[19]. Researchers [9, 17] have investigated program visualizations with a focus on code reading and code writing, and found it to be beneficial for students. For example, modeling the eye gaze of experts for novice programmers has shown promise in debugging tasks [21]. Live programming — where a program is designed and implemented in front of a class — has also been investigated as a pedagogical tool beneficial to code writing [15, 17]. There appears to be an absence of research on animating the presentation of a single textual representation of source code and how it may affect comprehension and carrying out of related tasks among novice programmers.

2.2 Methodology

2.2.1 Procedure. We tested 3 animation interventions with 16 novice programmers. Participants' eye-gaze data, as they interacted with the animations, was recorded using Gazepoint GP3 eye-tracker. The research team recruited participants from introductory programming courses as well as from participants of a summer research program for Masters students. After selection, some participants completed a survey that collected information about years of programming experience, currently enrolled programming courses, completed programming courses, and specifics for each programming language they have used. For other participants this information was collected verbally. All but three participants had less than four years of experience, and these three participants only piloted our first intervention (described below). Participants included students who had previously taken programming courses and students who were self-taught in programming.

A web-based programming environment was designed by the research team to display the animations. The user interface (UI) of the environment enabled participants to view the program and associated tasks, run the program, and debug by viewing error messages in the same screen. The programming tasks we tested included code summarization [8], syntax error correction, and logic error correction. *Code summarization* consisted of asking participants to summarize the code after watching an animation. *Syntax and logic error correction* are comparable to debugging tasks completed in other studies [12]. *Syntax errors* consisted of code errors that prevented source code execution such as a misspelled variable. *Logic errors* consisted of errors that caused code execution to produce inaccurate results such as an algorithm meant to sort in ascending order sorting in descending order. Participants were able

Figure 1: The interface of the live writing intervention as the animation reveals code in linear order. Participants were asked not to begin writing their summaries until after each animation completed playing.

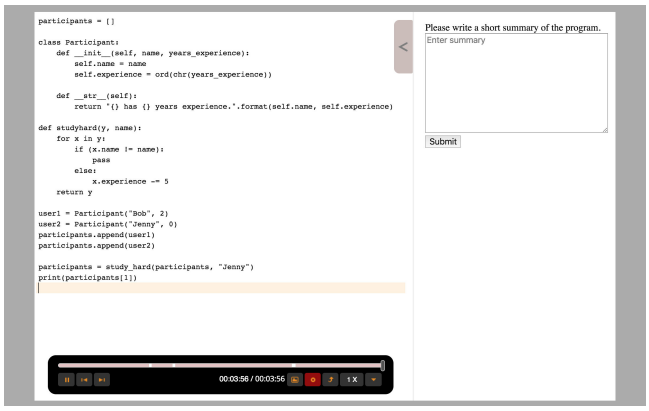

Figure 2: The live writing intervention after an animation completed. The presented code remained visible and editable during the summarization task.

to view, interact, and run a static representation of the code while completing all three tasks.

2.2.2 Line Highlighting Intervention. Based on the theoretical underpinning that making execution order explicit may be helpful to novice programmers [19], source code was presented with a white background and lines were highlighted in yellow in the order they would execute during run-time. In our initial iteration, lines were only highlighted during execution and then the background returned to white after execution. For our final iteration they stayed highlighted until the animation was completed and lines that executed multiple times, such as those within loops, became darker as the animation progressed. Six participants tested three variations of this intervention — variations included highlighting lines with code visible, blurry, and non visible.

2.2.3 Eye-Gaze Intervention. Source code was presented with a white background and a yellow dot. The interface was similar

Table 1: The operationalization of concepts for this study.

Concept	Definition	Operationalization (Metrics)
Novice Programmer	A programming student who has little to no previous experience [8].	Previous Coursework, Years of Experience, Self Reported Level of Knowledge [8]
Reading Approach	The approach/pattern taken when reading text [8].	Reading Patterns (Top-to-Bottom, Left-to-Right), Element Coverage [8]
Attentional Process	The processes of exploration and inspection which guide attention [14].	Ambient and Focal Attention measured with the K-Coefficient [14]

Figure 3: After completing and submitting their summaries, participants could view their next tasks, edit code, and execute the program.

Figure 4: Participants could view execution errors in the intervention interface while completing each task.

to Stein and Brennan's, which showed novice programmers the previously-recorded eye gaze of expert programmers represented as a yellow dot [21]. Similar to the line-highlighting intervention, this animation played in the order of program execution. Participants were instructed to follow a yellow dot overlaid onto the code editor. We tested the intervention with code visible, blurry, and non visible. We tested this intervention with 3 participants who also tested the line-highlighting intervention.

2.2.4 Live Writing Intervention. Participants were initially presented a blank integrated development environment (IDE) on the left and an area to enter a summary on the right. A Live Writing animation would then run where source code became visible similar to if it was being typed by the participant (Figures 1 and 2). Live Writing is a technology that records the typing of code allowing it to be viewed remotely and asynchronously[11]. After the animation played, the interface became a working IDE where participants could execute code, view errors, and complete tasks (Figures 3 and 4).

Three similar code blocks were created. The lengths of the blocks ranged from 23 to 27 lines. Each code block included a for-loop, a class, and a method. A fourth block was created and used for a warm-up task for all participants. This ensured participants were familiar with the interface controls, the tasks, and the code elements they would encounter. The code blocks were designed so all expressions and statements would execute when run (without errors).

After the warm-up, participants were presented each code block with one of three treatments: static code (included as a control), an animation of code being typed in the order it would execute, or an animation of code being typed in linear order (lines typed from top-to-bottom). The order and combination of each code block and treatment was randomized across participants. Ten participants completed pilots of the Live Writing Intervention.

2.2.5 Research Questions. The research questions we focus on in this paper are:

(1) The effect of the animated presentation of source code on a novice programmer's reading approach and attentional processes.
(2) Comparison of the effects of different interventions on reading approach and attentional processes.

2.3 Results

2.3.1 Element Coverage. Element coverage is defined as the fraction of words fixated on. [8]. We used this metric to explore differences in how participants interacted with the source code between tasks and during the intervention. We found a notable difference when comparing measures taken during the consumption of the intervention and measures taken during the later tasks. Element coverage per minute was much higher when participants were consuming an animation than during a task. This is likely because all participants at least partially track code as it is typed.

Interestingly, we did not observe any clear effects when comparing element coverage measures across interventions. This could be due to individual differences among participants, and also partly due to the small sample size. The only observed difference was between the participant group who completed the debugging task after viewing the non-linear Live Writing Intervention and the

groups who either were presented with static code or received the linear treatment.

For all three interventions, we examined the reading approach of our participants. We operationalized reading approach using the same measures defined by Busjahn et al [8]. This included element coverage, vertical next text (saccades that stay on same line or move one line below), vertical later text (saccades that stay on same line or move to any line below), horizontal later text (saccades that move right on same line), regression rate (saccades that move to previous lines), and line regression rate (saccades that move left on the same line) [8]. These indicators for reading approach were in line with logical explanations for animations (i.e. increased top-to-bottom line reading when viewing a linear animation of line typing).

2.3.2 Consuming Animations. For the Live Writing Intervention, participants were able to control the speed of the animation. Four participants viewed the animation at the speed we set (or slightly faster), and two viewed it at a much faster speed - this was expected behavior. We observed that two of the participants who played the animation quickly tracked the code as it was typed (Figure 5, left). When examining the eye tracking data we found that three of the participants who consumed the animation at a slower rate of speed were not just tracking the code as it was typed, but exploring all code visible at a given time (Figure 5, right). Concerning all three interventions, three participants stated they would likely skip viewing an animation if possible.

2.3.3 Attentional Processes after Consuming Animations. After observing participant behavior during the first two animation interventions, we included an analysis of the participants' attention processes with the Live Writing Intervention. Two modes of attentional processes — ambient and focal attention, which correspond to exploration and inspection behaviors respectively—operationalized by the K-coefficient—were analyzed (included in Table 1) [14]. The K-coefficient is calculated by taking the difference between zero-score values of the fixation durations and saccadic amplitudes. Positive values of the K-coefficient indicate focal viewing, while negative values indicate ambient viewing. The Live Writing treatment appears to increase ambient attentional processes when correcting logic errors, and increases focal attentional processes for activities like code summarization and syntax correction. This seems to suggest that novice programmers tend to scan the code for logic errors, but read the code more strategically for syntax errors and code summarization.

2.4 Discussion

2.4.1 Design of Animations. The eye-tracking data revealed that participants interacted with the animations differently. Our first two interventions - the line highlighting and eye-gaze animations - did not allow participants to control the speed of the animation. We received feedback from two participants that the animation was helpful and was played at a speed they felt was appropriate. Two participants felt the animation played too slowly. When we enabled participants to control the speed of animations we observed participants adjusting the speed up and down during the warm up task until settling on a speed. Participants preferred this control possibly increasing participant satisfaction with the animations.

Figure 5: Fixations of Participant 11 and 14. Participant 11 (left) tracked the animation at a high speed closely following the code as it became visible. Participant 14 (right) played the animation slowly, reading other parts of the computer code.

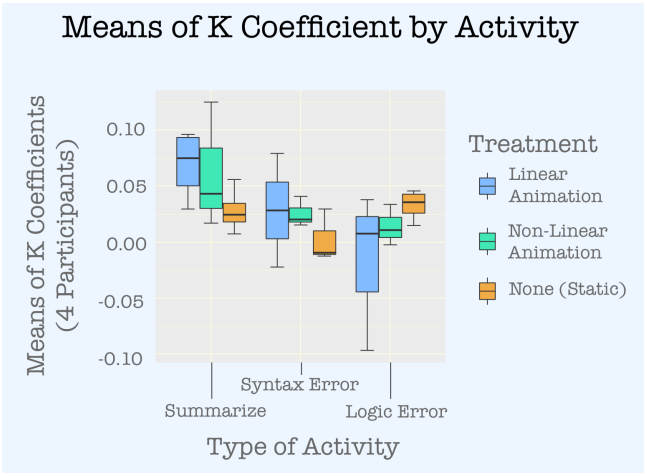

Figure 6: Live Writing appears to increase ambient attention when correcting logic errors while increasing focal attention with code summarization and correcting syntax errors.

Previous research has found that presentation speed can affect what concepts learners take away from an animation [13]. Although it is unclear if this would lead to better learning outcomes in this case, it is possible this control allowed participants who would have lost interest when the animation was playing slowly to maintain interest while still consuming the whole code block.

2.4.2 Effect on Reading Approach and Attentional Processes. Element coverage per minute was notably lower when debugging after viewing the non-linear Live Writing Intervention than the linear animation and with no intervention. It is possible this was because participants understood the code executed in a non-linear manner and participants read the code more efficiently. However, if this was the case, differences in vertical next text and vertical later text, which are indicative of the linearity of reading, would also be lower. This finding is important and should be further studied. If no differences exist between the effects of linear and non-linear animations, the total effect is from the animations alone.

We observed an increase in focal attention during debugging and summary tasks and an increase in ambient attention during logic tasks. The eye-tracking data revealed participants were reading the code during the code summary task, and inspecting the code for bugs during the debugging task, potentially leading to the increase in focal attention. The increase in ambient attention (which is also a decrease in focal attention) during the logic tasks aligns with participant behavior when they were lost and did not know where to look. It is yet to be seen what types of attentional processes will be most beneficial for novice programmers for various tasks making future work in this area all the more important.

3 FUTURE WORK

3.1 Limitations

It is possible that based on our small sample we observed anomalous behaviors that are not representative of novice programmers. It is also possible we missed observed outcomes that did not have a large effect size. The findings reported here should be viewed as preliminary and informative for future research rather than empirical and conclusive.

When completing our third invention, three participants piloted the interface with different code blocks and are excluded from these results. We experienced data collection issues with three other participants while using the eye tracker and their data was removed from the results.

3.2 Additional Metrics To Consider For Future Work

Additional qualitative and quantitative metrics exist that can support and enrich the findings in future studies. Two additional quantitative measures, story order and execution order, would provide insight into participant reading strategies at a global level [8]. As described by Busjahn et al., story order measures if code is read in linear order (like a book) whereas execution order measures if code is read in the order the lines of code would execute when run. Experts have been observed reading less linearly than novices [8]. In this study, we suspect that some participants were more likely to begin reading code in execution order after watching animations but would quickly fall back on reading in a more linear order. Previous research has found that "when the visual strategies of low-comprehenders were similar to those of high-comprehenders, the comprehension outcome of the low-comprehenders was poor" [4].

Qualitative data was collected through observations, the code summaries prepared by participants, and follow-up interviews. Analysis of this qualitative data would provide insight into participant mental models and overall comprehension. Similar to methodology laid out by Bendarik et al., a combined analysis of quantitative metrics and qualitative findings will provide insight into learning outcomes [4].

3.3 Future Work

The preliminary findings of this pilot study were in line with previous studies. The values of metrics measuring reading approach [8] and attentional processes [14] fell within the same ranges as previous studies adding face validity to our results. For example, our K-coefficient ranges were similar to the K-coefficient ranges Orlov observed for programmers that were coding in the programming language being studied less than one hour a week [14].

While we identified a possible effect on participants' attentional processes and element coverage when presenting live-written code, this finding should be further examined. A full scale study should examine the effects of each type of intervention explored in this study with a meta-analysis comparing the results of the three studies. This meta-analysis should focus on whether observed effects are the result of a specific intervention or from animations in general. Each study should focus on determining the role order-of-line animation plays in interventions.

ACKNOWLEDGMENTS

The authors would like to thank Sang Won Lee for providing support integrating Live Writing into our interface. This work is supported by the Institute of Museum and Library Services under Grant No.: RE-015-0086-15.

REFERENCES

[1] Ronald Baecker. 1998. Sorting out sorting: A case study of software visualization for teaching computer science. *Software visualization: Programming as a multimedia experience* 1 (1998), 369–381.

[2] Roman Bednarik, Teresa Busjahn, and Carsten Schulte. 2014. Eye movements in programming education: Analyzing the expert's gaze. In *Proc. 1st Int. Workshop.* Joensuu, Finland, 16–19.

[3] Roman Bednarik, Teresa Busjahn, and Sascha Schulte, Carsten; Tamm. 2016. *Eye Movements in Programming: Models to Data.*

[4] Roman Bednarik, Niko Myller, Erkki Sutinen, and Markku Tukiainen. 2006. Program visualization: Comparing eye-tracking patterns with comprehension summaries and performance. In *Proceedings of the 18th Annual Psychology of Programming Workshop.* 66–82.

[5] Roman Bednarik and Markku Tukiainen. 2004. Visual attention and representation switching in Java program debugging: A study using eye movement tracking. In *Proceedings of the 16th annual workshop of the Psychology of Programming Interest Group.* 159–169.

[6] Roman Bednarik and Markku Tukiainen. 2008. Temporal eye-tracking data: evolution of debugging strategies with multiple representations. In *Proceedings of the 2008 symposium on Eye tracking research & applications.* ACM, 99–102.

[7] C.; Tamm S.; Bednarik R. Busjahn, T.; Schulte. 2015. *Eye Movements in Programming Education II: Analyzing the Novice's Gaze* (2015).

[8] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *IEEE International Conference on Program Comprehension*, Vol. 2015-Augus. IEEE, 255–265. https://doi.org/10.1109/ICPC.2015.36

[9] Teresa Busjahn and Carsten Schulte. 2013. The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research - Koli Calling '13.* ACM Press, New York, New York, USA, 3–11. https://doi.org/10.1145/2526968.2526969

[10] Teresa Busjahn, Carsten Schulte, Bonita Sharif, Simon, Andrew Begel, Michael Hansen, Roman Bednarik, Paul Orlov, Petri Ihantola, Galina Shchekotova, and Maria Antropova. 2014. Eye Tracking in Computing Education *(ICER '14).* ACM, New York, NY, USA, 3–10. https://doi.org/10.1145/2632320.2632344

[11] Sang Won Lee and Georg Essl. 2015. Live writing: Asynchronous playback of live coding and writing. In *Proceedings of the International Conference on Live Coding.*

[12] Y T Lin, C C Wu, T Y Hou, Y C Lin, F Y Yang, and C H Chang. 2016. Tracking Students' Cognitive Processes During Program Debugging - An Eye-Movement Approach. *IEEE Transactions on Education* 59, 3 (aug 2016), 175–186. https://doi.org/10.1109/TE.2015.2487341

[13] Katja Meyer, Thorsten Rasch, and Wolfgang Schnotz. 2010. Effects of animation's speed of presentation on perceptual processing and learning. *Learning and Instruction* 20, 2 (apr 2010), 136–145. https://doi.org/10.1016/j.learninstruc.2009.02.016

[14] Pavel A Orlov. 2017. Ambient and Focal Attention During Source-code Comprehension. In *Eye Movements in Programming: Spring Academy 2017*, R.; Busjahn T.; Schulte C.; Vrzakova H.; Budde L. Tamm, S.; Bednarik (Ed.). Freie Universität, Berlin, Germany, 12–14.

[15] John Paxton. 2002. Live programming as a lecture technique. *Journal of Computing Sciences in Colleges* 18, 2 (2002), 51–56.

[16] Blaine A. Price, Ronald M. Baecker, and Ian S. Small. 1993. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages & Computing* 4, 3 (sep 1993), 211–266. https://doi.org/10.1006/jvlc.1993.1015

[17] Marc J. Rubin. 2013. The effectiveness of live-coding to teach introductory programming. *Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13* (2013), 651. https://doi.org/10.1145/2445196.2445388

[18] Juha Sorva. 2013. Notional Machines and Introductory Programming Education. *Trans. Comput. Educ.* 13, 2 (jul 2013), 8:1–8:31. https://doi.org/10.1145/2483710.2483713

[19] Juha Sorva, Ville Karavirta, and Lauri Malmi. 2013. A Review of Generic Program Visualization Systems for Introductory Programming Education. *ACM Transactions on Computing Education* 13, 4 (2013), 1–64. https://doi.org/10.1145/2490822

[20] John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price (Eds.). 1998. *Software visualization: Programming as a multimedia experience.* MIT Press, Cambridge, Massachusetts.

[21] Randy Stein and Susan E. Brennan. 2004. Another person's eye gaze as a cue in solving programming problems. *Proceedings of the 6th international conference on Multimodal interfaces* (2004), 9–15. https://doi.org/10.1145/1027933.1027936

[22] Sascha Tamm, Roman Bednarik, Teresa Busjahn, Carsten Schulte, Hana Vrzakova, and Lea Budde. 2017. *Eye Movements in Programming: Spring Academy 2017* (2017).

[23] Leelakrishna Yenigalla, Vinayak Sinha, Bonita Sharif, and Martha Crosby. 2016. How Novices Read Source Code in Introductory Courses on Programming: An Eye-Tracking Experiment. Springer, Cham, 120–131. https://doi.org/10.1007/978-3-319-39952-2_13