Extending iTrace-Visualize to Support Token-based Heatmaps and Region of Interest Scarf Plots for Source Code

Joshua A. C. Behler
Department of Computer Science
Kent State University
Kent, OH, USA
jbehler1@kent.edu

Giovanni Villalobos
Department of Computer Science
Kent State University
Kent, OH, USA
gvillalo@kent.edu

Julia Pangonis
Department of Computer Science
Kent State University
Kent, OH, USA
jpangoni@kent.edu

Bonita Sharif
School of Computing
University of Nebraska-Lincoln
Lincoln, Nebraska, USA
bsharif@unl.edu

Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent, Ohio, USA
jmaletic@kent.edu

***Abstract*—The iTrace Infrastructure is a suite of community eye-tracking tools that enables researchers to conduct eye-tracking studies on software projects in real development environments. The infrastructure consists of tools providing support for data gathering, post processing, and visualization. iTrace-Visualize provides researchers with a way to visualize gathered and post-processed eye-movement data. iTrace involves the analysis of more than just eye-movement data, and includes information gathered from the development environment and the source code. This work describes additions to iTrace-Visualize that provide researchers with visualizations of the gathered source code data. Specifically, a tokenized heatmap of the source code is presented, that shows source code tokens that are viewed the most. Additionally, a region of interest scarf plot that details the timeline of what parts of the code a participant views is added as a new feature. A small preliminary study comparing student and industry developers is presented to demonstrate the use of these tools.**
Demo Video: https://youtu.be/0iZcCC8CK94

***Keywords*—**eye tracking, visualization, heatmaps, pipeline

I. Introduction

The iTrace infrastructure [1-4] allows software engineering researchers to conduct eye-tracking studies within common development environments such as Eclipse and Visual Studio. iTrace-Visualize, a part of the infrastructure, allows for ways to visualize the eye-tracking information gathered during an eye-tracking session [5]. Because iTrace supports the dynamic stimulus of IDEs and other text editors, standard visualization techniques cannot be used. Visualizations such as heatmaps do not work well due to the stimulus changing, either from scrolling, changing files, or pop-up context windows appearing. In previous work, we explore various visualization methods, and ruled out typical methods because they tend to not work well over the dynamic medium of coding environments [5].

The first version of iTrace-Visualize supports gaze, fixation, saccade markup, and code line highlighting in gaze video playback with a fading display to keep the display clutter-free. In addition, support for video interpolation to support high-speed tracking [3] is built-in. After the first release, we received feedback from users that the visualizations offered by iTrace-Visualize were helpful for visualizing the eye-tracking information, but do not provide much insight into understanding which specific parts of the source code is viewed at more detail. To meet these concerns and support researchers in understanding what source code elements are examined and how participants navigate between them, we extend iTrace-Visualize to support two additional visualizations that are directly informed by the source code stimulus.

- **Tokenized Heatmaps:** Using the original source code and the fixations generated in iTrace-Toolkit, iTrace-Visualize creates a heatmap of how long tokens in the source code are viewed. The heatmaps have different options, allowing for the averaging and normalization of the data, as well as choosing whether the heatmap uses the duration or number of the fixations as the main metric of interest.
- **Region of Interest Scarf Plots:** iTrace-Visualize allows researchers to define chunks of lines as regions of interest per source code file, and then generates a scarf plot which details in a timeline how participants navigate across those regions.

The premise behind adding these additional visualizations is to aid researchers in further exploring their eye-tracking datasets for certain hypotheses based on their study goals. The tokenized heatmaps provide an option to see how long and/or how many times developers fixate on specific tokens in the code. This can help researchers identify if certain token types are harder to understand compared to others. For example, a variable that is poorly named leads developers to take longer to read and understand it. The region of interest scarf plots show the gaze patterns in source code such as usage of def/call chains where

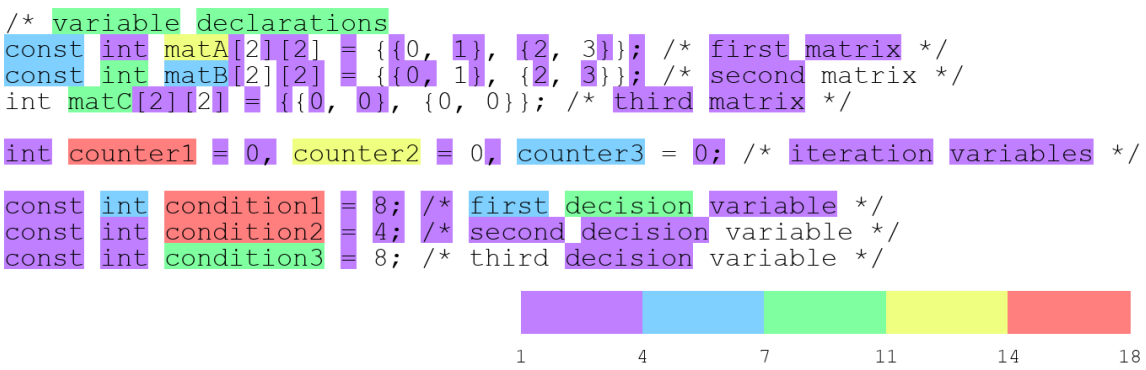

Figure 1: A segment of code turned into a tokenized heatmap. The legend is in number of fixations.

developers chase (navigate across) data values or follow control flow. One can do this by creating regions of interest for the definition and call areas of the code and plot the gazes across these areas via the region of interest scarf plots. It also allows researchers to develop their own custom regions to validate a hypothesis about how developers read. The plots generated can be used to provide evidence of gaze navigation observed.

II. Usage of iTrace-Visualize

The videos iTrace-Visualize produces provide a time replay of the eye-tracking data on the stimulus gathered from iTrace-Core and processed by iTrace-Toolkit [5]. iTrace-Visualize has been used to improve studies and other tools within the iTrace Infrastructure [6-11] We describe a few usage scenarios on how we have used iTrace-Visualize.

A. *Fixation Settings Adjustment*

iTrace-Toolkit provides users with three main fixation generation algorithms that aggregates recorded gaze information into fixations [4]. The three main algorithms are the Olsson Basic, I-VT, and I-DT algorithms [12, 13]. These algorithms have adjustable parameters, which users can change to better adapt to the specific needs of their study. However, it was difficult to know what settings would work best for a researcher from a glance. Because iTrace-Toolkit outputs its data into a SQLite database, viewing the generated fixations is difficult and tedious. Researchers would have to create their own visualization methods or perform arduous examinations of the database to determine if their settings worked well. Visualization helps with this process, as a researcher can generate a video that visualizes the fixations. Visualizing the fixations allows researchers to compare and contrast different fixation algorithms, or the same algorithm with different parameters. It also enables researchers to detect and address erroneous data originating from issues with their eye-tracking equipment. This capability not only prevents potential challenges but also minimizes time lost in the research process.

B. *Identifying Bugs*

The videos from iTrace-Visualize have been used within our own development team to identify issues with our other toolsets. Previously, iTrace-Toolkit had a semi-rare bug where extremely large fixations are generated under the I-DT algorithm. The I-DT filter requires the gazes of a fixation to be within a certain dispersion radius. Gazes that fall outside of this radius trigger the end of a fixation, and the calculation of a new one will begin. However, this algorithm had no limit on time, so long as the dispersion radius is maintained. Thus, if a participant looked off-screen for an extended period of time, and then looked back at roughly the same area they had left the screen from, an incredibly long fixation is recorded. These fixations are tens of seconds long, and interfere with studies where a participant has to look away from a screen occasionally. Using iTrace-Visualize, we were able to detect this bug and identify that the eye traveling off the screen is what triggered them. Researchers can then decide if they want to filter out such fixations. In our case, we fixed how the I-DT filter works to address the above concern.

C. *Going Beyond Video Playback*

The videos generated by iTrace-Visualize have many helpful attributes, but they did not solve all of the community's requests and needs for visualization. As stated in the introduction, a more specific source-code specific visualization was requested. This paper meets that need for providing two additional visualizations that directly aid in further analyzing eye tracking data. Some examples include using the tokenized heatmaps to determine which tokens are harder to comprehend by observing the frequency and duration of gazes. The region of interest scarf plots could help in understanding how novices and experts navigate code sections while fixing a bug for instance.

III. Tokenized Heatmap

In previous work, we initially dismissed heatmaps for visualizing the eye-tracking information generated iTrace due to the dynamic nature. We examined the works of Špakov and Miniotas [14], and Pfeffer and Memili [15] to see how heatmaps are currently being used with eye-tracking. Because iTrace allows for the recording of dynamic development environments, heatmaps could add a lot of visual clutter on top of an already complex development environment. Additionally, scrolling causes the heatmap to no longer align with the tokens in the source code. Visualizing the source code token information requires an entirely different methodology. The eye-tracking information is recorded in x and y pixel coordinates on the screen, which causes the main issue with visualization on a dynamic target. However, the source code information is recorded by saving the line and column from within the source code that the participant is examining. Using this, we can fully recreate the source code and visualize the individual tokens and characters that are viewed during the session.

Figure 2: Two tokenized heatmaps of the three novice (left) and three professional (right) developers. The heatmaps depict the average.

A. *Implementation*

iTrace-Visualize is implemented using Python and the Qt Python bindings. To recreate the source code for the heatmap, the OpenCV 4.8.1.78 and Pillow 10.3.8 imaging libraries are used. To separate the heatmap from the video replay from the first version of iTrace-Visualize, we add a tabbed view, where a user can switch between tabs depending on the type of visualization they want to perform. To create the heatmap, iTrace-Visualize converts the source code from the original study into an image. This image is sized depending on the size of the original source code, which means large files may result in very large images. As input, the heatmap generation takes an iTrace-Toolkit database, that has fixations calculated from the raw gaze data, and has the fixations mapped to source code tokens from the source code. The tool also requires a srcML [16, 17] archive of the source code to be able to recreate the code into an image and easily map the line and column information to individual tokens in the source code. srcML provides an XML representation of the source code's AST, and makes identifying tokens which are syntactically relevant easy.

B. *Generating a Heatmap*

After importing an iTrace-Toolkit database and a srcML archive, iTrace-Visualize populates a list with each session contained within the database. Selecting a database reveals the list of fixation runs – sets of fixations calculated with a specific fixation algorithm – that are associated with the selected session. Multiple sessions and fixation runs can be selected, allowing for batch processing. All selected fixation runs generate a single image per viewed file. For example, if two separate sessions had participants look at *main.cpp* and *calc.hpp*, generating heatmaps on both results in four images, two for each session from each file. The heatmaps use a standard visible light spectrum range to represent intensity, with violet representing the least viewed tokens and red representing the most viewed tokens. Each token viewed during a session is given a highlight of the corresponding color intensity. A user can specify the number of colors, from violet to purple, to use for their markup, with five being the default. These colors and their requisite values are drawn in as a legend at the bottom right of the image. Figure 1 demonstrates an example of the tokenized heatmap.

If a user wants to aggregate their data together instead of generating multiple images, they can average each selected fixation run together. This process will group all of the fixation data together per file, and will ignore participant and fixation run information. Only one image per source code file will be generated, showing the average heatmap of the file across all viewings. Two types of averaging are available, with either all fixation runs being added together, or all runs being normalized before adding, allowing a researcher to diminish the effect of outlier tokens. Additionally, iTrace-Visualize provides two ways to value each fixation for the heatmap. A user can either use the fixation count or the total fixation duration on a token.

IV. Region of Interest Scarf Plots

Alongside the individual tokens, researchers are interested in examining sections of the code and how developers transition between them during a task such as bug fixing for instance. These regions of interest provide insight into how developers read code and in what order. To aid researchers with this analysis, we provide the ability to generate region of interest scarf plots. These plots showcase which regions of the source code the participant viewed. The plots are generated using the *matplotlib* library [18]. Scarf plots are a popular way for researchers to visualize the viewing of regions of interest. We examined the *Alpscarf* tool by Yang et al. [19] and by Falzone et al. [20], who used scarf plots to visualize and analyze various search tasks on websites. These studies gave us a basis for how scarf plots are being used, and allow us to implement our scarf plots in adherence with community standards.

A. *Generating a Scarf Plot*

Like a heatmap, the region of interest scarf plots require an iTrace-Toolkit database as input. After importing the database, a list of sessions and fixations runs is shown, allowing a user to select one or more fixation runs for processing. Additionally, a user can specify the regions of interest on the source code.

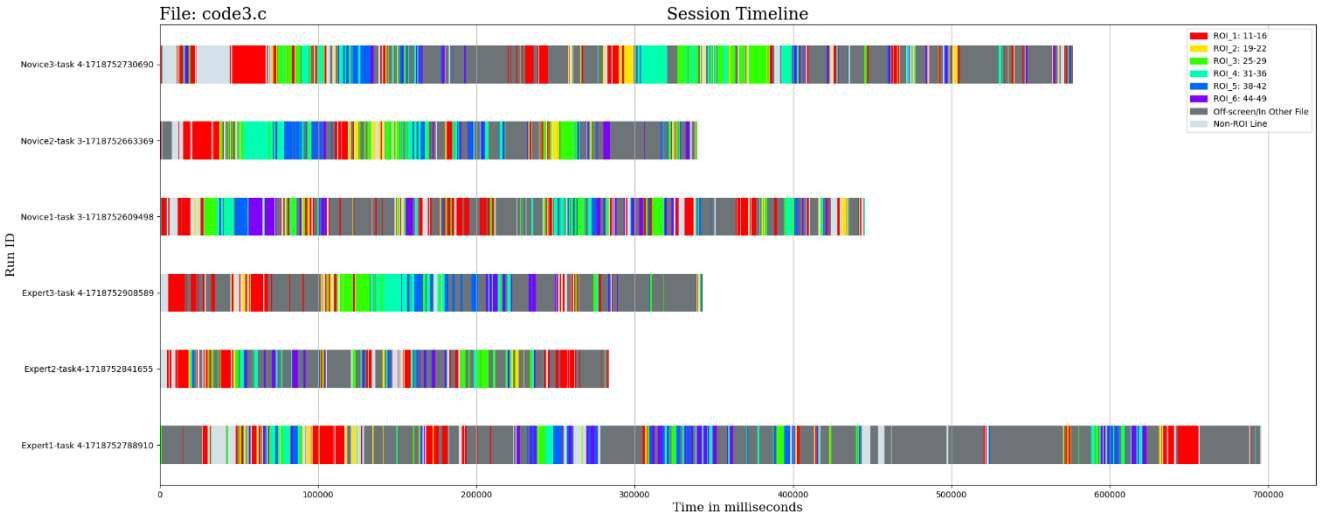

Figure 3: A region of interest scarf plot of all six developers. The top three (Novice 1-3) are the novice developers, and the bottom three (Expert 1-3) are the professional developers. The x-axis represents the milliseconds elapsed from the beginning of the recording session.

Target files can be added manually by a researcher, and then individual regions of interest can be specified. Regions of interest can be exported and imported by a researcher to save time if processing is done over multiple launches of the program. A researcher can specify a start and an end line for each region.

Generating the images will generate one image per file added to the list. For each image, each fixation run is examined for any fixations which targeted the corresponding file. If one is found, the fixation run will be plotted on the image. Like with the heatmaps, the colors will range from violet to red along the visible light spectrum. However, the colors hold no intensity meaning, and are automatically chosen to give each region of interest a specific color. Alongside those colors, two different shades of grey are plotted to represent fixations on lines not within a region of interest and fixations on other files or periods of times when the user was not looking at the screen. If a researcher defines no regions of interest, the graphs will still plot when the user was looking at the file and when they were looking at other files or were not looking at the screen. The graphs are plotted with respect to time, which additionally allows researchers to compare the length of the sessions.

V. Preliminary Case Study

We examine in a small pilot case study the differences between student and industry-level developers to demonstrate the new visualization techniques. Our data is taken from a separate currently ongoing research study measuring the efficacy of code review techniques. From this dataset, we randomly pull data from three undergraduate student developers and three professional developers who currently work in industry. The participants are instructed to view the source code and examine it for any issues or formatting errors. Any found errors were to be recorded in a separate text file in the code editor. For clarification, we are not making any concrete claims about the nature of novice and professional developers in this work - rather, we are using them as an example to explain an example use of iTrace-Visualize. Using this data, we run the novices and the industry developers through iTrace-Visualize and generate the tokenized heatmaps and the scarf plots shown in Figures Figure **2** and Figure **3** respectively. The generated graphs, both combined and individual, are available in our artifact[1].

Examining the heatmaps, we see from a glance that the industry developers spent more time looking at the comment near the top of the file than the novices, but the novices looked at the comments above the statements within the functions. On the scarf plots, each region of interest is a significant section of the source code shown in the heatmaps. The red region represents the declarations at the top of the main, the yellow is the if statement after, and the remainder are the for loops within the else. The dark gray represents when the user was either looking off-screen or at the response sheet. Looking at the scarf plots, we see that the professionals switch between regions of the code and their response sheet more often than the novices, who tend to examine the code for longer. The professionals also did not view ROI 3 as much as the novices, which contains a loop that sets the initial contents of the array.

VI. Conclusions and Future Work

iTrace-Visualize offers researchers a quick and easy way to understand eye-tracking data on source code and provides qualitative analysis for statistical results. Through our own use, we found that using iTrace-Visualize allows us to draw conclusions about our results, which we can then further investigate. iTrace-Visualize is available for download from our website at https://www.i-trace.org. In the future, we plan to expand the visualizations supported based on further feedback from our community. One possible addition is incorporating an optional timelapse to the tokenized heatmap generation to see the heatmap change over time. Augmenting the scarf plots similar to Alpscarfs [19] to support the display of revisits and longer stimuli is another venue for future improvements.

[1] https://osf.io/xw8bf/?view_only=5f86c328426e481ca48f512b19be4469

References

[1] Bonita Sharif and Jonathan I. Maletic, "iTrace: Overcoming the Limitations of Short Code Examples in Eye Tracking Experiments," presented at the 32nd IEEE International Conference on Software Maintenance and Evolution (ICSME), Oct. 2016, pp. 647–647. doi: 10.1109/ICSME.2016.61.

[2] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, "iTrace: eye tracking infrastructure for development environments," in *10th ACM Symposium on Eye tracking Research and Applications*, Warsaw, Poland, Jun. 2018, p. 3. doi: 10.1145/3204493.3208343.

[3] V. Zyrianov *et al.*, "Deja Vu: semantics-aware recording and replay of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks—methodology and analyses," *Empir Software Eng*, vol. 27, no. 7, p. 168, Dec. 2022, doi: 10.1007/s10664-022-10209-3.

[4] J. Behler, P. Weston, D. T. Guarnera, B. Sharif, and J. I. Maletic, "iTrace-Toolkit: A Pipeline for Analyzing Eye-Tracking Data of Software Engineering Studies," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, May 2023, pp. 46–50. doi: 10.1109/ICSE-Companion58688.2023.00022.

[5] J. Behler, G. Chiudioni, A. Ely, J. Pangonia, B. Sharif, and J. I. Maletic, "iTrace-Visualize: Visualizing Eye-Tracking Data for Software Engineering Studies," in *2023 11th IEEE Working Conference of Software Visualization*, Aug. 2023.

[6] K. Park, P. Weill-Tessier, N. C. C. Brown, B. Sharif, N. Jensen, and M. Kölling, "An eye tracking study assessing the impact of background styling in code editors on novice programmers' code understanding," in *Proceedings of the 2023 ACM Conference on International Computing Education Research - Volume 1*, in ICER '23, vol. 1. New York, NY, USA: Association for Computing Machinery, Sep. 2023, pp. 444–463. doi: 10.1145/3568813.3600133.

[7] J. A. Saddler *et al.*, "Studying Developer Reading Behavior on Stack Overflow during API Summarization Tasks," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb. 2020, pp. 195–205. doi: 10.1109/SANER48275.2020.9054848.

[8] C. S. Peterson, J. A. Saddler, N. M. Halavick, and B. Sharif, "A Gaze-Based Exploratory Study on the Information Seeking Behavior of Developers on Stack Overflow," in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, in CHI EA '19. New York, NY, USA: Association for Computing Machinery, May 2019, pp. 1–6. doi: 10.1145/3290607.3312801.

[9] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz, "Eye gaze and interaction contexts for change tasks – Observations and potential," *Journal of Systems and Software*, vol. 128, pp. 252–266, Jun. 2017, doi: 10.1016/j.jss.2016.03.030.

[10] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *Proceedings of the 26th Conference on Program Comprehension - ICPC '18*, Gothenburg, Sweden: ACM Press, 2018, pp. 286–296. doi: 10.1145/3196321.3196347.

[11] I. Bertram, J. Hong, Y. Huang, W. Weimer, and Z. Sharafi, "Trustworthiness Perceptions in Code Review: An Eye-tracking Study," in *ESEM '20: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 5-7, 2020*, M. T. Baldassarre, F. Lanubile, M. Kalinowski, and F. Sarro, Eds., ACM, 2020, p. 31:1-31:6. doi: 10.1145/3382494.3422164.

[12] D. D. Salvucci and J. H. Goldberg, "Identifying Fixations and Saccades in Eye-tracking Protocols," in *2000 Symposium on Eye Tracking Research & Applications*, in ETRA '00. Palm Beach Gardens, Florida, USA: ACM, Nov. 2000, pp. 71–78. doi: 10.1145/355017.355028.

[13] P. Olsson, "Real-time and Offline Filters for Eye Tracking," Masters Thesis, KTH Electrical Engineering, Stockholm, Sweden, 2007. Accessed: Jun. 21, 2019. [Online]. Available: https://pdfs.semanticscholar.org/4167/7844556582adc68a5a14dbb1cea0b28d9016.pdf

[14] O. Špakov and D. Miniotas, "Visualization of Eye Gaze Data using Heat Maps," *Elektronika Ir Elektrotechnika*, vol. 74 No. 2, pp. 55–58.

[15] T. Pfeiffer and C. Memili, "Model-based real-time visualization of realistic three-dimensional heat maps for mobile eye tracking and eye tracking in virtual reality," in *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, in ETRA '16. New York, NY, USA: Association for Computing Machinery, Mar. 2016, pp. 95–102. doi: 10.1145/2857491.2857541.

[16] M. L. Collard, M. J. Decker, and J. I. Maletic, "Lightweight Transformation and Fact Extraction with the srcML Toolkit," in *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, Williamsburg, Virginia, USA: IEEE, Sep. 2011, pp. 173–184. doi: 10.1109/SCAM.2011.19.

[17] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration," in *29th IEEE International Conference on Software Maintenance (ICSM)*, 2013, pp. 516–519. doi: 10.1109/ICSM.2013.85.

[18] "Matplotlib — Visualization with Python." Accessed: Jun. 16, 2024. [Online]. Available: https://matplotlib.org/

[19] C.-K. Yang and C. Wacharamanotham, "Alpscarf: Augmenting Scarf Plots for Exploring Temporal Gaze Patterns," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, Montreal QC Canada: ACM, Apr. 2018, pp. 1–6. doi: 10.1145/3170427.3188490.

[20] K. Falzone, S. Lemonnier, T. Grébert, and C. Bastien, "Using Scarf Plots to Visualize Moment-to-Moment Visual Search Behavior on Websites," in *Adjunct Proceedings of the 34th Conference on l'Interaction Humain-Machine*, in IHM '23 Adjunct. New York, NY, USA: Association for Computing Machinery, Jul. 2023, pp. 1–8. doi: 10.1145/3577590.3589604.