INVITED CONTENT

Editor: **Giuliano Antoniol**
Polytechnique Montréal

Editor: **Steve Counsell**
Brunell University

Editor: **Phillip Laplante**
Pennsylvania State University

Tracking Developers' Eyes in the IDE

Bonita Sharif, Timothy Shaffer, Jenna Wise, and Jonathan I. Maletic

FOR MANY YEARS, researchers have used eye trackers to study how people comprehend visual stimuli.[1] Eye movements are essential to cognitive processes because they focus a person's visual attention on the parts of a visible stimulus that are processed by the brain. Visual attention triggers the cognitive processes required for comprehension.

Eye movement is a proxy for cognitive effort and lets us determine whether what someone is looking at is difficult to understand. Cognitive psychology has used eye trackers to study how people comprehend words, prose, pictures, and diagrams. Computer scientists have used eye-tracking devices to study how people interact with GUIs and webpages. One goal of such investigations is to learn what constitutes a good human–computer interface so that we can design better ones.

Here, we look at how eye tracking can help researchers investigate how software developers perform tasks in IDEs, with the goal of improving developer support for those tasks. In particular, we introduce iTrace, our eye-tracking tool.

Eye-Tracking Basics

Eye trackers comprise a hardware and a software component They determine the x and y coordinates of where a person is looking at a screen. To do this, cameras record the user's eye movements, at a given sampling rate and a given accuracy specific to the eye tracker. Eye trackers are typically accurate to 0.5 degrees (a 0.25-in diameter) on the screen.

Two types of eye movements are *fixations* and *saccades*. A fixation is the stabilization of eyes on an object of interest. Saccades are quick eye movements from one location to the next. A *scan path* is a directed path formed by saccades between fixations. The general consensus in the eye-tracking research community is that visual-information processing occurs only during fixations.[1]

Figure 1 shows eye gazes on source code and a UML class diagram. Circles indicate fixations. The bigger a circle's radius, the longer the user looked at that point. The number in a circle indicates the order in which the fixation occurred.

What's Missing?

For a fixed stimulus, such as an image that's the same size as the screen, mapping the eye gaze coordinates to the location a person is looking at is relatively straightforward geometry. Changes to the stimulus (screen), such as scroll-

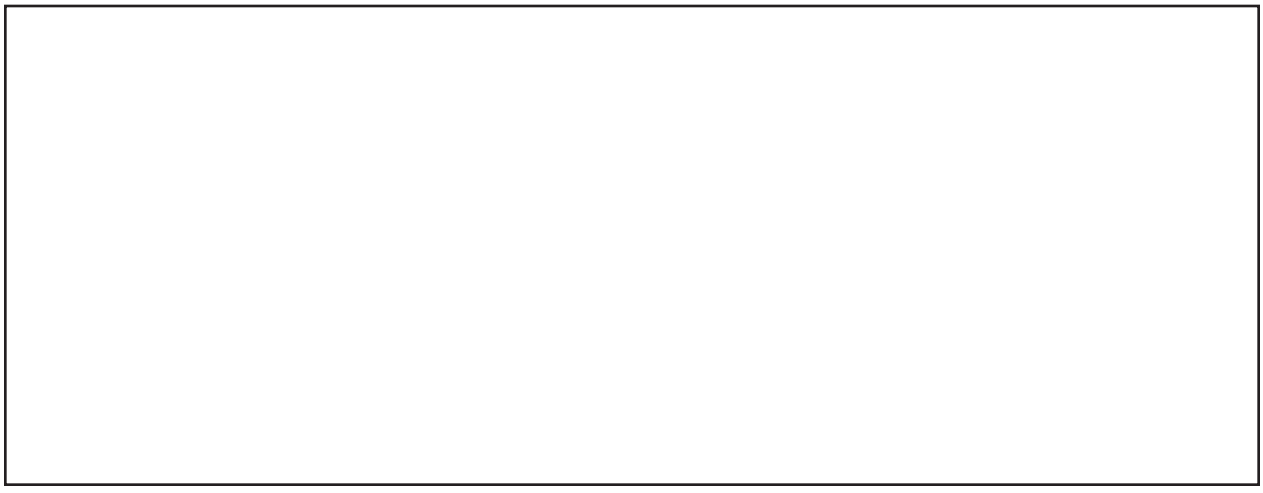

FIGURE 1. Scan paths on (a) snippets of source code and (b) a UML class diagram. Each circle indicates a fixation—the stabilization of eyes on an object of interest. The bigger a circle's radius, the longer the user looked at that point. The number in a circle indicates the order in which the fixation occurred.

ing through a large file or opening multiple files, complicate the problem because the stimulus is moving or constantly changing. Commercial eye-tracking software provides limited support for scrolling; it requires a fixed stimulus that just happens to be longer than what fits on the screen. It provides no support for tracking a person interactively using an editor or switching between files.

Basically, existing systems don't keep track of what line in which file is on the screen (that is, being viewed). Software engineering researchers are using eye trackers to study how engineers comprehend and develop software.[2] Unfortunately, eye trackers' fixed-stimulus limitations are particularly problematic for studying software developers plying their trade. Although a fixed-sized stimulus is sufficient to study how people read a sentence (or a few lines of source code), it's wholly inadequate to study how programmers try to comprehend an entire software system.

Software developers deal with many artifacts, such as requirements documents, design documents and diagrams, source code, test cases, defect data, build data, release data, project management data, and code inspection documents. No support exists to track eye movements on these artifacts seamlessly as the developer works. Remember that an eye tracker only reports the x and y coordinates of where a person is looking; it isn't aware of what's at that position.

After collecting the eye-tracking data, the researcher must manually create areas of interest in the stimulus—in our case, around lines or specific elements of the source code or UML symbols. At this point, the researcher maps the developer's eye gaze to the areas of interest in the lines of code or parts of the diagram. However, this process is tedious and can't scale to thousands of lines of code or large UML class diagrams. Mapping eye gaze to certain areas of interest isn't automatic. Although this process works fine for simple small visual stimuli, it's inadequate for investigating real-world work environments.

Making Eye Tracking Implicit in IDEs

To address the limitations we just discussed, we developed iTrace,[3] a plugin to the Eclipse IDE. iTrace interfaces with an eye tracker and the IDE elements to capture eye gaze on software artifacts and map them to their semantic meaning on the fly. It automatically maps each gaze of a developer to the source code elements he or she is looking at. It also keeps track of the viewed source code's syntactic structure. The developer can switch between artifacts while working, and iTrace distinguishes the context of what he or she is examining.

iTrace lets developers work naturally on tasks such as reading and writing code, fixing bugs, impact analysis, and feature location. It doesn't prohibit them from scroll-

ing or switching between files. It saves eye gaze session data in standard JSON (JavaScript Object Notation) and XML files following the examined artifact's syntax and semantic structure.

iTrace is extensible enough to track any software artifact, including ad hoc documents relevant to each industrial firm. As long as users can open the files in Eclipse, iTrace can track those files. We've released an iTrace prototype (https://github.com/YsuSERESL/iTrace) as open source under GPL (GNU General Public License). The same concepts are applicable to Visual Studio; we're looking into that.

Toward More Realistic Research

Paige Rodeghero and her colleagues studied eye tracking on small code snippets to determine where developers look while they summarize methods.[4] The participants spent more time looking at method signatures than method invocations and more time looking at invocations than control flow. With iTrace, researchers could conduct such a study in a familiar work environment. That is, they'd see the method along with all the rest of the code, instead of in isolation. This might produce different results.

Impact on Program Comprehension

We used iTrace to investigate 22 developers performing realistic change tasks on JabRef, an open source bibliography manager.[5] To investigate the benefits of iTrace-generated data, we compared eye-tracking data with Mylyn interaction history data (edits and selections made), both of which are gathered simultaneously. Compared to the interaction data, iTrace captured more contextual data on source code elements and, what's more important, captured different aspects of developer activity. Other findings were that the developers were interested in data flow inside a method and preferred navigating between methods in close proximity instead of following call relationships.

Impact on Software Traceability

Software traceability is the ability to explicitly link all related artifacts across the software development and maintenance life cycle. We conducted a pilot study to determine whether we could use eye gaze to infer traceability links from bug reports to source code.[6] We devised an algorithm that gives a higher weight to source code elements seen later in the session. The rationale is that when developers start a session, they aren't sure what's important. But as the session ends, they focus only on source code entities they think are task-related. The results indicate that this approach achieves strong recall for traceability links when developers accurately perform bug localization tasks.

Large-Scale Adoption

Eye trackers have significantly decreased in cost and improved in usability over the years. Major eye-tracking vendors (Tobii, SR Research, and SensoMotoric Instruments) have tailored their product packages to many budget ranges ($100–$40,000). This hardware price drop, along with iTrace, provides exciting new areas of research in understanding how developers work. It's not far-fetched to imagine a future in which eye trackers will be common on computers.

iTrace could transform the way we study how developers build software. Software engineering researchers will be able to conduct large-scale realistic eye-tracking studies seamlessly in a software development environment using many different artifacts. For developers to accept eye-tracking technology as part of their daily routine, it must be integrated into the workflows and development environments with which they're comfortable.

The data generated from developer sessions will help us build better tools that are integrated in the IDE to support tasks such as code recommendations based on eye gazes. These recommendations will depend highly on how an individual works. This is similar to how search is contextualized to each individual's profile and history. Recommendations can also be individualized to each developer's eye context.

This implicit eye tracking will also directly impact other software engineering tasks such as code summarization, code reviewing, and software traceability. The more we understand how developers work, the better we can support them by reducing information overload, improving defect prediction and code summarization, or providing automatic recommendations when they're stuck or frustrated.[7] With recorded eye gaze sessions, researchers will be able to determine all the locations a developer examined to address a task. This will pave the way to further improve IDEs to support developers in various software engineering tasks.

References

1. K. Rayner, "Eye Movements in Reading and Information Processing: 20 Years of Research," *Psycho-*

logical Bull., vol. 124, no. 3, 1998, pp. 372–422.
2. Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A Systematic Literature Review on the Usage of Eye Tracking in Software Engineering," *Information and Software Technology*, Nov. 2015, pp. 79–107.
3. T. Shaffer et al., "iTrace: Enabling Eye Tracking on Software Artifacts within the IDE to Support Software Engineering Tasks," *Proc. 2015 10th Joint Meeting Foundations of Software Eng.* (ESEC/FSE 15), 2015, pp. 954–957.
4. P. Rodeghero et al., "Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers," *Proc. 36th IEEE/ACM Int'l Conf. Software Eng.* (ICSE 14), 2014, pp. 390–401.
5. K. Kevic et al., "Tracing Software Developers' Eyes and Interactions for Change Tasks," *Proc. 2015 10th Joint Meeting Foundations of Software Eng.* (ESEC/FSE 15), 2015, pp. 202–213.
6. B. Walters et al., "Capturing Software Traceability Links from Developers' Eye Gazes," *Proc. 22nd Int'l Conf. Program Comprehension* (ICPC 14), 2014, pp. 201–204.
7. S. Müller and T. Fritz, "Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress," *Proc. IEEE/ACM 37th Int'l Conf. Software Eng.* (ICSE 15), 2015, pp. 688–699.

BONITA SHARIF is an assistant professor in Youngstown State University's Department of Computer Science and Information Systems. Contact her at bsharif@ysu.edu.

TIMOTHY SHAFFER is a PhD student in the University of Notre Dame's Department of Computer Science and Engineering. Contact him at tshaffe1@nd.edu.

JENNA WISE is an undergraduate student in mathematics and computer science at Youngstown State University. Contact her at jlwise@student.ysu.edu.

JONATHAN I. MALETIC is a professor in Kent State University's Department of Computer Science. Contact him at jmaletic@kent.edu.