Practical Eye Tracking with iTrace

Bonita Sharif *, Cole S. Peterson*, Drew T. Guarnera†, Corey A. Bryant†, Zachary Buchanan†,
Vlas Zyrianov†, and Jonathan I. Maletic†
*Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska, USA 68588
†Department of Computer Science, Kent State University, Kent, Ohio, USA 44242
Emails: bsharif@unl.edu, cole.scott.peterson@huskers.unl.edu, {dguarner, cbryan20, zbuchana, vzyriano, jmaletic}@kent.edu

***Abstract*—The evolution and effort in designing and implementing iTrace, an infrastructure for integrating eye tracking into developer environments, is presented. The goal is to make eye tracking practical for various stakeholders in software engineering namely researchers, practitioners, and educators. An overview of iTrace and the general process involved in conducting an eye tracking study with human subjects using iTrace is presented in this tool demo paper. Upcoming features and ongoing plans for community involvement are also presented.**

***Index Terms*—eye tracking, practical solution, integrated development environments, program comprehension, empirical studies**

I. Introduction

Eye tracking is gaining popularity in the software engineering community as a method to understand how software developers work [1]. Here, we provide an overview of *iTrace*, our eye tracking infrastructure. *iTrace* makes conducting eye tracking studies accessible to software engineering researchers as it alleviates many of the pain points involved in conducting eye tracking studies. One major obstacle prior to *iTrace* is that program comprehension studies can realistically only be conducted on short code snippets. This is due to software limitations in determining what elements of the code are being observed if the view of the code editor changes during scrolling or switching files. Much tedious post processing is involved including video processing of screen recordings and manually mapping gazes to semantic elements in a long video recording. *iTrace* overcomes this limitation by keeping track of the information being looked at when the screen contents change and then automatically maps those coordinates to the syntactic features in the code during a post-processing phase. Researchers can then focus on more interesting analysis and drastically reduce the time spent post-processing.

The goal of this tool demo paper is two-fold. We first present the current state of *iTrace* as it has evolved over its early development years. The second goal is to introduce the general set of steps to follow when using *iTrace* during an eye tracking study. For additional details on tool setup and usage we direct the reader to our web portal at http://www.i-trace.org.

II. iTrace History and Overview

iTrace is a community infrastructure for performing eye tracking studies within an integrated development environment (IDE) such Visual Studio or Eclipse to more closely resemble realistic software development conditions. Code is not written in isolation within the IDE and to better fit the process of how developers work, we are also working on supporting eye tracking within Chrome via a plugin named *iTrace-Chrome*. This extends eye tracking capabilities to websites such as Stack Overflow, Bugzilla and GitHub. Such infrastructure facilitates new directions for conducting research studies.

The first iTrace prototype was created in 2012 where it existed only as an Eclipse plugin. In 2018, we introduced a new and completely refactored version of iTrace [2]. The application is now split into several different plugins to easily facilitate extension and reuse. The main plugins are *iTrace-Core*, *iTrace-VisualStudio*, *iTrace-Eclipse*, and *iTrace-Chrome*. The system is designed in a manner that facilitates extensions for other IDE platforms such as Visual Studio Code and IntelliJ. *iTrace-Core* is responsible for interfacing with the eye tracker, managing session data, and broadcasting the gaze coordinates to the plugins. Plugins are created for a specific IDE and use the gaze coordinates sent by the core application in conjunction with APIs provided by the IDE to map the screen position of the user's gaze to an interface element. If the IDE element contains source code, a gaze can be mapped down to the line and column of the source code element. All mappings are collected and written to an XML file where line and column mappings are used with *srcML* (www.srcML.org) to process the eye tracking data and gain useful insights into developer's eye movements. We are also working on support tools such as *iTrace-PostProcessing* and *iTrace-Analysis* that will be released in the near future. The complete set of tools are hosted on GitHub at https://github.com/iTrace-Dev.

New Features: Since the 2018 refactored version [2], we added the ability for *iTrace-Core* to communicate with plugins via TCP socket or WebSocket connections. This allows plugins to be developed for several additional platforms such as Google Chrome and VS Code. Socket settings are user configurable to avoid port conflicts, and *iTrace* supports multiple plugin clients running simultaneously. Along with enhancements for plugin communication, *iTrace* has support for more trackers including the GazePoint GP3 series tracker alongside the Tobii Pro and Tobii 4C (with Pro upgrade). See Figure 1 for the *iTrace-Core* interface. To better assist in the analysis of eye tracking data generated by *iTrace*, we have greatly extended the post-processing features of the infrastructure to allow for all gaze data and analysis to be recorded in a single SQLite database. Moreover, this data need not be from only one participant's session, but encompass a complete study package, facilitating researchers in creating

978-1-7281-2243-4/19/$31.00 ©2019 IEEE
DOI 10.1109/EMIP.2019.00015

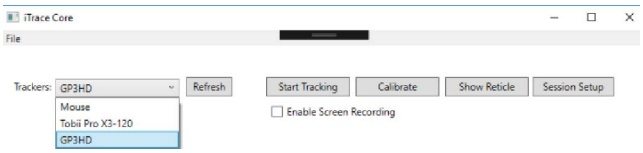

Fig. 1. *iTrace-Core* Interface showing the eye trackers currently connected. The GazePoint GP3 tracker and the Tobii X3-120 trackers are visible. The mouse tracker can also be used as a proxy for testing when an eye tracker is not readily available. Recording the screen is also an option for replaying gaze overlays.

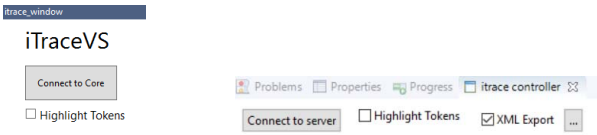

Fig. 2. Plugin Views. Visual Studio's view is shown to the left with Eclipse's view shown to the right. We provide feature parity between plugins. Currently, both can connect to the server and highlight tokens in code given where the developer is looking.

artifacts for their accepted papers. It also allows for the easy exchange of data between collaborators. Any analysis of the data external to *iTrace* can use standard SQL commands or exported for later manipulation in external applications.

Upcoming Features: A current limitation of *iTrace* is its inability to map gaze data accurately as source code is edited during an eye tracking session. The ability to study software development in the context of editing, such as bug fixing or refactoring is of great research significance. This is a challenging problem, and the *iTrace* team is actively exploring solutions to support eye tracking for live source code editing. Another upcoming feature involves recording the screen during the eye tracking sessions. This feature allows session playback with an overlay of eye fixations and saccades as they occur - a useful aide for qualitative analysis. Additional features for data validation and correction are also being designed.

III. Running Your Eye-tracking Study

In order to run an eye tracking study, you first need to install *iTrace* and one of the plugins and have access to a supported eye tracker. We have successfully used *iTrace* in two published studies [3, 4].

- Session setup: The first step is to set up the session in *iTrace-Core*. There are four fields that must be populated: Study Name, Researcher Name, Participant ID, and Data Directory. It is important to be consistent with the Study Name and Data Directory and to use distinct Participant IDs as that makes processing easier after the study.
- Calibration: Next, the participant goes through a 9-point calibration screen. Results are reported visually and written to file for later use.
- Plugin Setup: In each plugin, there is a button to establish a connection to Core. See Figure 2. Once this connection is established, the core will alert the plugin of any new recording session that starts.
- Start Tracking: Once the above steps are successfully completed, tracking is started from *iTrace-Core*. With the Core running, data from the eye tracker is transmitted to the plugins.

IV. Community

We are actively working to help develop a community around *iTrace*. Our website at http://www.i-trace.org/ features downloads for *iTrace-Core* and our post-processing tool for detecting fixations. The source code repositories for our Eclipse and Visual Studio plugins are open source (with installers for each being released in the near future). All available tools and feature documentation to help with installation and usage will be posted to our web portal and YouTube channel. Our site also tracks analytics to better help us gauge interest in certain content and tools that have been released. We plan to make all of our GitHub repositories public not only to receive feedback in the form of bugs and feature requests, but also to encourage community contribution and engagement. Presently, feedback and issues are reported via email (itracedev@gmail.com) on our contact page. We will be expanding this to allow for feature requests, bug reports, and questions to submitted directly from the site. A community workshop session has been scheduled at ICSE 2019.

V. Conclusions

This tool paper reports on the current state of *iTrace* including the background and motivation of why it is important. The effort of community building is presented along with a set of steps on how to start using *iTrace* to conduct a study. As part of our future work, we plan on adding short tutorials and videos to increase the adoption of this framework among software engineering researchers.

Acknowledgements

This work is supported by the National Science Foundation under grant numbers CCF 18-55756, CCF 15-53573, and CNS 17-30307/30181.

References

[1] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng, "A survey on the usage of eye-tracking in computer programming," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 5:1–5:58, Jan. 2018.

[2] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, "itrace: Eye tracking infrastructure for development environments," in *Proc. of ETRA*, 2018, pp. 105:1–105:3.

[3] N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic, "Developer reading behavior while summarizing java methods : Size and context matters," in *Proceedings of the 41st International Conference on Software Engineering (ICSE)*, 2019.

[4] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, T. Fritz, and D. C. Shepherd, "Tracing software developers eyes and interactions for change tasks," *Proc. of ESEC/FSE*, 2015.