iTrace: Eye Tracking Infrastructure for Development Environments

Drew T. Guarnera
Kent State University
Department of Computer Science
Kent, Ohio, USA
dguarner@kent.edu

Corey A. Bryant
Kent State University
Department of Computer Science
Kent, Ohio, USA
cbryan20@kent.edu

Ashwin Mishra
Youngstown State University
Department of CSIS
Youngstown, Ohio, USA
amishra01@student.ysu.edu

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

Bonita Sharif
Youngstown State University
Department of CSIS
Youngstown, Ohio, USA
bsharif@ysu.edu

ABSTRACT

The paper presents *iTrace*, an eye tracking infrastructure, that enables eye tracking in development environments such as Visual Studio and Eclipse. Software developers work with software that is comprised of numerous source code files. This requires frequent switching between project artifacts during program understanding or debugging activities. Additionally, the amount of content contained within each artifact can be quite large and require scrolling or navigation of the content. Current approaches to eye tracking are meant for fixed stimuli and struggle to capture context during these activities. *iTrace* overcomes these limitations allowing developers to work in realistic settings during an eye tracking study. The *iTrace* architecture is presented along with several use cases of where it can be used by researchers. A short video demonstration is available at https://youtu.be/AmrLWgw4OEs

CCS CONCEPTS

• **Human-centered computing**;

KEYWORDS

eye tracking infrastructure, integrated development environments

ACM Reference Format:
Drew T. Guarnera, Corey A. Bryant, Ashwin Mishra, Jonathan I. Maletic, and Bonita Sharif. 2018. iTrace: Eye Tracking Infrastructure for Development Environments. In *ETRA '18: 2018 Symposium on Eye Tracking Research and Applications, June 14–17, 2018, Warsaw, Poland.* ACM, New York, NY, USA, Article 4, 3 pages. https://doi.org/10.1145/3204493.3208343

1 INTRODUCTION

Eye trackers are a critical research tool in understanding how software developers comprehend source code and other visual stimuli. The software engineering community has been conducting studies with biometric devices including eye trackers [Obaidellah et al. 2018; Sharafi et al. 2015] to understand how developers read and understand software artifacts such as source code. An eye tracker gives researchers a unique view into thought processes otherwise not observed. Software developers spend the vast majority of their development time performing program comprehension activities [Minelli et al. 2015]. Current hardware and software offered by eye tracking vendors support fixed stimuli with tedious post processing to map eye gaze to areas of interest (AOIs). The tedious post processing even on simple fixed stimuli is clearly inadequate for software artifacts such as source code. Software developers not only work with multiple source code files but each of these files typically is comprised of thousands of lines of code. They constantly flip through multiple files while reading and debugging code. Since source code files are really large, we cannot feasibly draw AOIs around each element to map gaze data to those regions. In addition, source code is both semantically and syntactically rich and structured different from natural language text [Busjahn et al. 2015]. To accurately capture the context of what developers are looking at, an eye tracking infrastructure is needed that can handle dynamic screen actions such as file switching, code folding, and content scrolling.

The *iTrace* infrastructure is a novel solution that seeks to solve the above mentioned problem by integrating eye tracking into developer work environments to support conducting large-scale eye tracking studies. *iTrace* is extensible and customizable to support gaze data on multiple software artifacts such as text files, html documents, and source code to name a few. After a quick calibration, *iTrace* runs uninterrupted in the background within the developer environment, recording developer eye movements while they are working. These gazes will be automatically mapped to specific code elements via a post processing module. For more information, visit the *iTrace* website at http://www.i-trace.org.

2 ARCHITECTURE AND DESIGN

The architecture for *iTrace* (significantly extended from an earlier prototype [Shaffer et al. 2015]) uses a central core application dedicated to setting up a study session and interacting with any attached eye trackers (See Figure 1). The core is responsible for interfacing with the eye tracker, calibration, session configuration, and broadcasting gaze points to plugins. Plugins are developed for specific development environments and communicate with the core to receive gaze points along with bookkeeping metadata. When a session is finished the core and plugin will both produce a set of recorded data about the study in XML format. The response records are joined by the *session time* field in each gaze response record. The plugins are responsible for recording line and column information that map the eye gaze to text or interface elements within the integrated development environment (IDE) and recorded during the study. *iTrace* is designed with the goal of supporting additional eye trackers and plugins in the future. Feature parity will be maintained between plugins.

Instead of using real-time AST information derived from the IDE, a post processing module maps the line and column information saved by plugins to syntactic elements in source code via the *srcML* format [Collard et al. 2011, 2013; Collard and Maletic 2016]. This occurs as part of a post processing phase where source code is converted to the *srcML* format and together with gaze data is mapped to semantically meaningful elements via an automated tool chain. *iTrace* maps elements down to the token being viewed. For example, in the declaration `float totalCost;` we get eye gaze mapped on the data type `float` and the identifier `totalCost` as separate entities. Besides getting tokens mapped automatically, *srcML* is also able to give us information about the type of syntactic element (method call, function call among many others). This makes is simple for a researcher to query the final generated gaze files to ask specific questions about developer behavior.

Finally, we have the option of running various fixation filters to generate fixations from the combined raw gaze files tagged with AST information. The final generated data can then be used towards some functional goal such as determining navigation behavior.

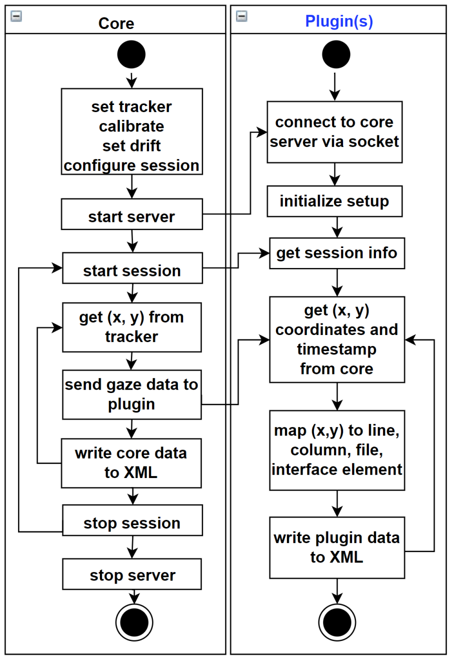

Figure 1: Overview of the iTrace architecture

3 USE CASES

We envision *iTrace* helping researchers, practitioners, and educators. We present a few of the many use cases [Sharif et al. 2016] below.

Program Comprehension. *iTrace* can map gaze data to both the content of source code at a syntactic level, and development environment views. This can improve understanding about what source code elements help facilitate program comprehension and what environment presentations are most often used. The information can lead to identifying areas for code refactorings or enhancing tool support to assist developers with the construction of mental models [Altmann 2001; Ko et al. 2006; Soloway and Ehrlich 1984]. *iTrace* was successfully used for the purpose of determining the types of elements developers look at while fixing bugs [Kevic et al. 2017, 2015]. They showed that eye movement data is richer and more detailed than interaction data. Clark et al. developed a tool to visualize the corresponding gaze data [Clark and Sharif 2017].

Software Traceability. *iTrace* supports context switching between multiple source code artifacts allowing for gaze data to help identify related source code elements. With support for both IDEs and web browsers, artifacts can be bug reports, documentation, etc. *iTrace* was used in two studies on traceability [Sharif et al. 2017; Walters et al. 2014] where an algorithm was developed to discover links based on elements looked at. They showed eye gaze to be a feasible method to discover traceability links, in particular, the hidden links that are not easily generated from information retrieval systems.

Interviews. *iTrace* can be used by interviewers to see how their candidates perform at job interviews. Imagine a scenario where the interviewer gives the candidate a large open source system with a bug to fix. Using *iTrace*, the interviewer will be able to see via eye gaze, the strategies the candidate used during problem solving. Perhaps the process to solve the problem is more important than the final answer. Educators can benefit in similar ways where they learn how students read and understand code.

Expertise Prediction. Analysis of how developers interact and read code may provide insight into a developer's level of experience. With future support planned for tracking source code editing during a study, this has the potential to provide even more insight into skill level.

4 CONCLUSIONS AND FUTURE WORK

The paper presents a novel infrastructure that allows eye tracking to be used in a dynamic environment that closely resembles real world development. A set of utility tools will be developed as part of future work that tie into the *iTrace* infrastructure. It will be extensible enough so others can contribute to this tool set. The goal is to support other eye tracking researchers, developers, and educators in learning more about developer gaze patterns.

ACKNOWLEDGMENTS

This work has been funded in part by the National Science Foundation under Grant Numbers CCF 1553573 and CNS 17-30307/30181.

REFERENCES

Erik M. Altmann. 2001. Near-term memory in programming: a simulation-based analysis. *International Journal of Human Computer Studies* 54, 2 (2001), 189–210.

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 *Proceedings of 22th International Conference on Program Comprehension (ICPC '15)*.

Benjamin Clark and Bonita Sharif. 2017. iTraceVis: Visualizing Eye Movement Data Within Eclipse. In *IEEE Working Conference on Software Visualization, VISSOFT 2017, Shanghai, China, September 18-19, 2017*. 22–32. https://doi.org/10.1109/VISSOFT.2017.30

Michael L. Collard, Michael John Decker, and Jonathan I. Maletic. 2011. Lightweight Transformation and Fact Extraction with the srcML Toolkit. In *11th IEEE Working Conference on Source Code Analysis and Manipulation, SCAM 2011, Williamsburg, VA, USA, September 25-26, 2011*. 173–184. https://doi.org/10.1109/SCAM.2011.19

Michael L. Collard, Michael John Decker, and Jonathan I. Maletic. 2013. srcML: An Infrastructure for the Exploration, Analysis, and Manipulation of Source Code: A Tool Demonstration. In *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. 516–519. https://doi.org/10.1109/ICSM.2013.85

Michael L. Collard and Jonathan I. Maletic. 2016. srcML 1.0: Explore, Analyze, and Manipulate Source Code. In *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. 649. https://doi.org/10.1109/ICSME.2016.36

Katja Kevic, Braden Walters, Timothy Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. 2017. Eye gaze and interaction contexts for change tasks - Observations and potential. *Journal of Systems and Software* 128 (2017), 252–266. https://doi.org/10.1016/j.jss.2016.03.030

Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, Thomas Fritz, and David C. Shepherd. 2015. Tracing software developers eyes and interactions for change tasks. *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (2015).

Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32 (2006), 971–987. https://doi.org/10.1109/TSE.2006.116

Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I know what you did last summer: an investigation of how developers spend their time. In *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC 2015, Florence/Firenze, Italy, May 16-24, 2015*. 25–35. https://doi.org/10.1109/ICPC.2015.12

Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. 2018. A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Comput. Surv.* 51, 1, Article 5 (Jan. 2018), 58 pages. https://doi.org/10.1145/3145904

Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks. (2015), 954–957. https://doi.org/10.1145/2786805.2803188

Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. 2015. A Systematic Literature Review on the Usage of Eye-tracking in Software Engineering. *Information and Software Technology (IST)* (2015).

Bonita Sharif, Benjamin Clark, and Jonathan I. Maletic. 2016. Studying developer gaze to empower software engineering research and practice. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*. 940–943. https://doi.org/10.1145/2950290.2983988

Bonita Sharif, John Meinken, Timothy Shaffer, and Huzefa H. Kagdi. 2017. Eye movements in software traceability link recovery. *Empirical Software Engineering* 22, 3 (2017), 1063–1102. https://doi.org/10.1007/s10664-016-9486-9

Elliot Soloway and Kate Ehrlich. 1984. Empirical Studies of Programming Knowledge. *IEEE Trans. Software Eng.* 10, 5 (1984), 595–609.

Braden Walters, Timothy Shaffer, Bonita Sharif, and Huzefa H. Kagdi. 2014. Capturing software traceability links from developers' eye gazes. In *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*. 201–204. https://doi.org/10.1145/2597008.2597795