iTrace: Eye Tracking Infrastructure for Development Environments

Extended Abstract

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

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 [11, 13] 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 [10]. 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 [2]. 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 [12]) 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
We envision web browsers, artifacts can be bug reports, other source code files, float we get eye gaze mapped on the data type with the goal of supporting additional eye trackers and plugins in the future. The final generated data can then be used towards the educator would be interested in how students read whereas environment presentations are most often used. This information can lead to identifying areas for code improvements (refactorings) or enhancing tool support to assist developers with the construction of mental models [1, 9, 16]. iTrace was successfully used for the purpose of determining the types of elements developers look at while fixing bugs [7, 8]. 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 [3].

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. This information can lead to identifying areas for code improvements (refactorings) or enhancing tool support to assist developers with the construction of mental models [1, 9, 16]. iTrace was successfully used for the purpose of determining the types of elements developers look at while fixing bugs [7, 8]. 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 [3].

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, other source code files, documentation, etc. iTrace was used in two studies on traceability [15, 17] where an algorithm was developed to discover links based on elements developers 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 because they involve tacit knowledge that developers know but are not explicitly documented in the source code.

3 USE CASES

We envision iTrace helping researchers, practitioners, and educators. The educator would be interested in how students read whereas the practitioner could be interested in navigation strategies for bug fixing. We present a few of the many use cases [14] 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. This information can lead to identifying areas for code improvements (refactorings) or enhancing tool support to assist developers with the construction of mental models [1, 9, 16]. iTrace was successfully used for the purpose of determining the types of elements developers look at while fixing bugs [7, 8]. 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 [3].

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, other source code files, documentation, etc. iTrace was used in two studies on traceability [15, 17] where an algorithm was developed to discover links based on elements developers 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 because they involve tacit knowledge that developers know but are not explicitly documented in the source code.

3 USE CASES

We envision iTrace helping researchers, practitioners, and educators. The educator would be interested in how students read whereas the practitioner could be interested in navigation strategies for bug fixing. We present a few of the many use cases [14] 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. This information can lead to identifying areas for code improvements (refactorings) or enhancing tool support to assist developers with the construction of mental models [1, 9, 16]. iTrace was successfully used for the purpose of determining the types of elements developers look at while fixing bugs [7, 8]. 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 [3].

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, other source code files, documentation, etc. iTrace was used in two studies on traceability [15, 17] where an algorithm was developed to discover links based on elements developers 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 because they involve tacit knowledge that developers know but are not explicitly documented in the source code.

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


