EyeDE: Gaze-enhanced Software Development Environments

Hartmut Glücker
University of Regensburg
93040 Regensburg, Germany
hartmut.gluecker@student.ur.de

Felix Raab
University of Regensburg
93040 Regensburg, Germany
felix.raab@ur.de

Florian Echtler
University of Regensburg
93040 Regensburg, Germany
florian.echtler@ur.de

Christian Wolff
University of Regensburg
93040 Regensburg, Germany
christian.wolff@ur.de

CHI 2014, Apr 26 - May 01 2014, Toronto, ON, Canada
ACM 978-1-4503-2474-8/14/04.
http://dx.doi.org/10.1145/2559206.2581217

Abstract
This paper introduces *EyeDE*, a prototypical system enabling gaze interaction for assistance in integrated development environments (IDE). By utilizing an eye tracking device, we have enhanced an IDE prototype with gaze-controlled interaction methods for source code navigation. A qualitative evaluation shows that users welcome the ability to quickly look up documentation or to jump to method declarations just by looking at triggers placed in the code. Although inaccuracies inherent in eye tracking technology and discomforting sitting positions for users impede successful implementation of more advanced IDE features, the interaction paradigm appears to be acceptable within the software development context and seems promising as eye tracking technology is being further improved.

Author Keywords
Eye tracking; IDE; Software development environment; Source code interaction

ACM Classification Keywords
H.5.2. User Interfaces: Input devices and strategies; Interaction styles; Screen design.

Introduction

Integrated development environments (IDEs) offer a vast number of tools for developer assistance. However, IDEs often lack ease of use with regard to how these tools can be found or activated – which may be part of the reason why the *Eclipse* developer team has recently focused on usability-related enhancements for their development environment [3]. This finding indicates that there is a general need for IDEs to support software developers with using the tools in a more productive manner and thus constitutes the motivation for this work.

Furthermore, programmers suffering from repetitive strain injury (RSI) may benefit from multimodal development environment setups and possibly even retain their ability to produce code – which is another reason why research should also consider accessibility improvements in the context of software development. This work presents *EyeDE*, a prototype utilizing an eye tracking device to assist programmers during the development process. We want to simplify reading and navigating source code by appropriately reacting to gaze. We propose a design that enables quick and natural navigation through gaze-controlled menus in IDEs. The interaction is designed to support hands-free navigation, not only enabling handicapped developers to browse the source code, but also preventing able-bodied developers from repeatedly moving their hands between the keyboard and the mouse for certain tasks.

Related Work

Enhancing development environments

Introducing a speech recognition algorithm specifically designed for understanding Java commands, Begel and Graham [1] successfully integrate speech-based code synthesis into the *Eclipse* IDE, providing the missing counterpart to the reading-focused approach presented in this work: combining the two modalities may enable handicapped developers to both navigate through menus by gaze and to actually create code by speech.

Sharif & Kagdi show that eye tracking can be helpful for discovering software traceability issues [7]. More recently, Walters et al. have presented *iTrace*, an eye tracking-enhanced *Eclipse* plug-in that is used for creating links in software design artifacts like UML diagrams [9].

Current eye tracking applications

Biedert et al. [2] enrich the reading experience of digital books with eye tracking technology. Analyzing the readers' gaze, the project *Text 2.0* determines their current reading position in order to display contextual information such as figures or images in novels. The application also offers a "skimming mode" that grays out stop words when the user is glancing over the text.

A large body of eye-tracking-related work is focused on making applications accessible for users who are unable to use their hands. For instance, van der Kamp [5] successfully implements a multimodal painting application that utilizes both speech recognition and gaze control to let users draw without using their hands.

The "Midas Touch" problem with eye tracking

Due to the fact that it is hard for a system to distinguish between a user's intention of triggering functionality and the aim of exploring the user interface (UI) with their gaze, using eye tracking technology has a major design tradeoff that is generally referred to as

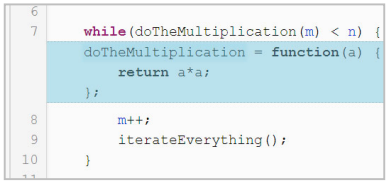

Figure 1. Using local expand to view a method's content.

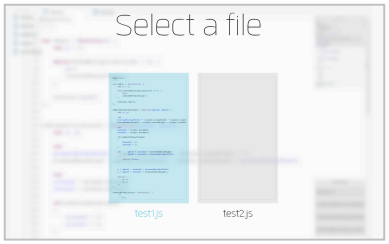

Figure 2. Users can switch files using their gaze.

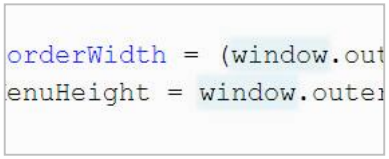

Figure 3. Further occurrences of a token are highlighted upon gaze.

Figure 4. Contextual menu activated by focusing the method call.

the "Midas Touch" issue [4]. It arises when the visual exploration of the UI activates gaze-based functionality without the user's consent. While there are techniques to avoid this situation, they usually tend to result in a less natural way of interaction. Most ideas to alleviate the Midas Touch problem are derived from two main mechanisms originally proposed by Jacob [4]. He suggests that a gaze-based event should only be triggered when:

- the gaze remains stationary in the area of interest (AOI) for a predefined dwell time threshold, and/or
- a key is pressed on the keyboard while the user is focusing the AOI with their gaze.

Since these methods are non-exclusive, Jacob suggests combining both. The dwell time threshold can be perceived as slow and exhausting whereas a key press can speed the selection up for experienced users. Zhang & MacKenzie have shown that eye/click interaction can be faster than mouse-based positioning and interacting in a GUI [10]. As far as gaze-based menu controls are concerned, Urbina et al. [9] present a novel alternative to Jacob's suggestions: Using a pie-like menu structure, one can trigger the selection of a "slice" by moving the gaze from the center across its border. Even though this method was determined as faster and more accurate, the test subjects did not consider it as intuitive as selection based on dwell time.

Application Overview

In this section we describe the features the *EyeDE* prototype offers, the technical foundations of the prototype and the reasoning behind the design decisions we made.

Application features

EyeDE is a prototypical code editor featuring only a small subset of the capabilities of an IDE. For the time being, our work focuses on hands-free navigation for reading and understanding code. Therefore we have included the following features:

- jump to method declaration (resp. jump back)
- documentation lookup
- locally expand method body (resp. collapse method body, see Figure 1)
- switch between files (see Figure 2)
- highlight occurrences of variable/method names (see Figure 3)
- outline / mini-map navigation
- expand / minimize the project explorer

We consider the first four features invasive, that is, they disrupt the workflow by either bringing up new elements in the user interface or performing an unexpected action (e.g. jumping to a method declaration when only reading a method call). The latter three features are triggered without the need for additional confirmation since they are either subtle, expectable and/or easily reversible.

A menu for switching files appears when the user's gaze focuses the file names in the tab bar (see Figure 2). Looking at a method call will yield a menu with three items (see Figure 4) for either jumping to the method declaration, looking up its *JavaDoc* comment or to locally expand the method to see its body "inline". By gazing at the mini-map or methods listed in the outline, the application scrolls to the corresponding position in the code. Whenever a variable or method is being focused for a short time span, the token and all

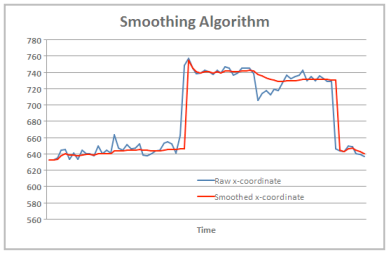

Figure 5. Results of the smoothing algorithm proposed by [6].

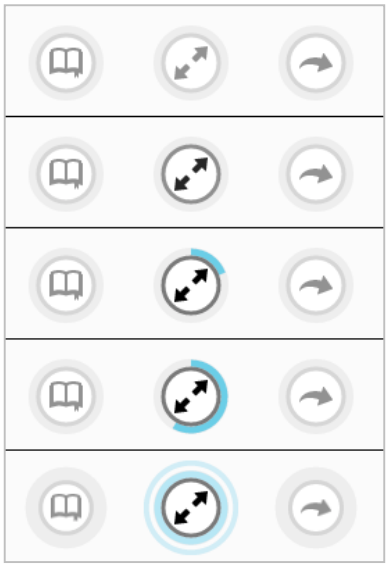

Figure 6. The "local expand" bubble is being activated. Upon activation, it emits a wave while the other bubbles fade out.

of its other occurrences within the document are subtly highlighted (see Figure 3). Finally, looking at the project explorer (that is collapsed to save more space for the actual source code) will expand it to its full width to make the file tree readable. Looking away returns it to its previous state.

Application architecture

The application is built upon the following components:

- eye tracking server (with periphery)
- fixation detection server
- client application with code editor

The client will only receive gaze coordinates from the fixation detection server and is not coupled to the heuristics that are being used in the detection algorithm. *EyeDE* utilizes the *RED-m* eye tracking device by *SensoMotoric Instruments* (SMI).[1] The *iView X* software constitutes the eye tracking server component. Its API can be accessed using the *iView X* SDK which is available for C#, C++ and Python.

The fixation detection server functions as "man in the middle", communicating with both the eye tracking server and the code editor client. Via the *iView X* API, it regularly polls the current gaze data, uses a weighted-mean algorithm [6] to reduce jitter, and sends the smoothed data (see Figure 5) to the JavaScript client using a WebSocket connection established with the *Tornado* package for Python. [2]

[1] http://www.smivision.com/en/gaze-and-eye-tracking-systems/products/redm.html

[2] http://www.tornadoweb.org

Our primary research goal is to examine both the usefulness and usability of eye tracking in the software development context. Within the scope of this prototype, a browser-based JavaScript client will suffice as long as the interface still "feels" like a code editor. For this purpose, the open source software *CodeMirror*[3] is used. This component supports various basic features available in code editors such as syntax highlighting, code indentation and a programming API.

Application design

In a visual environment controlled by gazes, it is important to avoid distractions as they may cause users to focus on them either voluntarily or accidentally. Since the user interface consists of several objects reacting to gaze, such distractions could potentially result in unwanted triggering of actions. Besides the display of source code, the *EyeDE* UI therefore only utilizes light gray tints with low contrast (see Figure 7), with the only highlighting color being a light blue for displaying the status of certain actions.

We regard the cautious use of colors as important so that our animated contextual menus do not draw too much attention to themselves while still being noticeable. Dwelling over certain UI objects opens the contextual menu with selection options that are designed to resemble "bubbles". When the user gazes at such a bubble, a blue arc will keep filling its outer border (see Figure 6) until the user averts the gaze from the bubble. Once the filling animation is complete, the corresponding action is triggered.

[3] http://www.codemirror.net

Figure 7. *EyeDE* UI overview.

The contextual menu adds another layer to the dwell-time-based selection mechanism: At worst, exploring the UI can cause the bubble menus to pop up or trigger minor actions like occurrence highlighting. Invasive actions, however, can only be triggered by explicitly gazing at a bubble.

Age	Amount
older than 40	1
30 - 40	5
24 - 29	1
18 - 23	1

Table 1. Participant age distribution.

Evaluation

In order to test the interaction paradigm, we invited eight participants from different software development backgrounds to take a 15-minute test and ask them to share their subjective impressions with us after using *EyeDE*. Out of the eight participants, four wore glasses and a fifth was cross-eyed. Except for one wearer of glasses, none of the visually impaired passed the initial tutorial section of the application due to severely skewed calibration results. The remaining four participants had to re-calibrate the device a few times to gain gaze control over the application. This issue caused some frustration and had a large impact on the user experience.

The remaining four participants were instructed to understand what an intentionally obfuscated JavaScript program did without navigating with the mouse or keyboard. In a pilot study, major issues in the interaction design were detected and addressed. Participants who did not use all of the *EyeDE*'s features to accomplish the task were then asked to try each of the remaining features.

During the interview part of the tests, we questioned the participants about their experience using *EyeDE* and how it could be improved. All in all, they described their experience as interesting and intuitive. We determined that they sometimes felt the need to skip the dwell time – especially when the gaze data fluctuated due to misrecognition. Therefore, as suggested by Jacob [4], we finally implemented a mixture of both dwell-time- and key-press-based selection mechanism. When the *Alt key* is pressed while the target bubble is being focused, it will be activated immediately, skipping the remaining dwell time. With respect to the misrecognition problem, we will implement a "magnetic aiming aid" for future releases.

Some participants remarked that, while reading code, they would expect to be able to lean back and view the code from afar. Gaze controls would be valuable in that scenario as the mouse or keyboard are not required to navigate through files. However, the eye tracker required the users to stay seated in a distance of approximately 50 to 70 centimeters to the device.

Future Work and Conclusion

We have presented *EyeDE*, a prototype of a gaze-controlled software development environment. Our system enables users to perform code navigation tasks in an editor by gazing at custom triggers.

Despite the issues inherent in current eye technology, our preliminary qualitative study shows that this interaction method may enhance the software development experience. Future work includes the following fields:

Gaze-enhanced IDEs

Our evaluation shows that using gaze control for code navigation appears to be perceived as natural. In future work, we will broaden the scope of this study, both in terms of adding more gaze-based IDE features to the software and conducting further usability tests.

Multimodal IDEs

As we pointed out earlier, combining and integrating both speech recognition and eye tracking into IDEs may enable handicapped software developers to make full use of a full-value development environment.

References

[1] Begel, A., & Graham, S. L. An assessment of a speech-based programming environment. In *Proc. VL/HCC 2006*, 116-120.

[2] Biedert, R., Buscher, G., Schwarz, S., Hees, J., Dengel, A. Text 2.0. In *CHI 2010 Extended Abstracts,* 4003-4008.

[3] Hou, D., & Wang, Y. An empirical analysis of the evolution of user-visible features in an integrated development environment. In *Proc. CASCON 2009,* 122-135.

[4] Jacob, R. J. Eye movement-based human-computer interaction techniques: Toward non-command interfaces. *Advances in human-computer interaction*, *4*, 1993, 151-190.

[5] van der Kamp, J. Gaze-Based Paint Program with Voice Recognition, 2010 (Dissertation).

[6] Kumar, M. GUIDe saccade detection and smoothing algorithm. *TR Stanford CSTR*, *3*, 2007.

[7] Sharif, B., & Kagdi, H. On the use of eye tracking in software traceability. In *Proc. TEFSE 2011*, 67-70.

[8] Urbina, M. H., Lorenz, M., Huckauf, A. Pies with EYEs: the limits of hierarchical pie menus in gaze control. In *Proc. ETRA 2010,* 93-96.

[9] Walters, B.; Falcone, M.; Shibble, A.; Sharif, B. Towards an eye-tracking enabled IDE for software traceability tasks. In *Proc. TEFSE 2013,* 51-54.

[10] Zhang, X., MacKenzie, I. S. Evaluating Eye Tracking with ISO 9241 - part 9. In *Proc. HCI 2007,* 779-788