EMIP Toolkit: A Python Library for Customized Post-processing of the Eye Movements in Programming Dataset

Naser Al Madi
Colby College
Waterville, Maine, USA
nmadi@kent.edu

Drew T. Guarnera
The College of Wooster
Wooster, Ohio, USA
dguarnera@wooster.edu

Bonita Sharif
University of Nebraska–Lincoln
Lincoln, Nebraska, USA
bsharif@unl.edu

Jonathan I. Maletic
Kent State University
Kent, Ohio, USA
jmaletic@kent.edu

ABSTRACT

The use of eye tracking in the study of program comprehension in software engineering allows researchers to gain a better understanding of the strategies and processes applied by programmers. Despite the large number of eye tracking studies in software engineering, very few datasets are publicly available. The existence of the large Eye Movements in Programming Dataset (EMIP) opens the door for new studies and makes reproducibility of existing research easier. In this paper, a Python library (the EMIP Toolkit) for customized post-processing of the EMIP dataset is presented. The toolkit is specifically designed to make using the EMIP dataset easier and more accessible. It implements features for fixation detection and correction, trial visualization, source code lexical data enrichment, and mapping fixation data over areas of interest. In addition to the toolkit, a filtered token-level dataset with scored recording quality is presented for all Java trials (accounting for 95.8% of the data) in the EMIP dataset.

CCS CONCEPTS

• **Software and its engineering → Software organization and properties**; • **Human-centered computing** → *Human computer interaction (HCI).*

KEYWORDS

eye-movement, programming, source code, eye tracking, toolkit, library

ACM Reference Format:
Naser Al Madi, Drew T. Guarnera, Bonita Sharif, and Jonathan I. Maletic. 2021. EMIP Toolkit: A Python Library for Customized Post-processing of the Eye Movements in Programming Dataset. In *ETRA '21: 2021 Symposium on Eye Tracking Research and Applications (ETRA '21 Short Papers), May 25–27, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3448018.3457425

1 INTRODUCTION

Eye tracking is gaining popularity as a tool in human-oriented software engineering, providing evidence on attention and the cognitive processes of programmers [Obaidellah et al. 2018]. This popularity is evident by surveying 31 papers in the field in 2015 [Sharafi et al. 2015] and 63 papers in 2018 [Obaidellah et al. 2018]. A practical guide on how to conduct studies in software engineering was also published in 2020 [Sharafi et al. 2020].

One of the first papers to use eye tracking in software engineering research is [Crosby and Stelovsky 1990] in 1990, yet the use of eye tracking did not become a well-established research method in software engineering until 2010-2012 [Lai et al. 2013]. Today, the use of eye tracking in software engineering research can be categorized into five areas: program comprehension [Aschwanden and Crosby 2006; Bednarik and Tukiainen 2006; Binkley et al. 2013; Busjahn et al. 2011; Crosby and Stelovsky 1990; Duru et al. 2013; Maalej et al. 2014; Sharif and Maletic 2010; Turner et al. 2014], debugging [Bednarik and Tukiainen 2007; Hejmady and Narayanan 2012; Romero et al. 2002], model comprehension [De Smet et al. 2014; Guéhéneuc 2006; Jeanmart et al. 2009; Porras and Guéhéneuc 2010; Sharif and Maletic 2010; Soh et al. 2012; Yusuf et al. 2007], collaborative programming [Sharma et al. 2015; Stein and Brennan 2004], and traceability [Ali et al. 2012; Sharif et al. 2017; Walters et al. 2014]. The use of eye tracking in the study of program comprehension in software engineering allows researchers to gain a better understanding of the strategies and processes applied by programmers [Madi et al. 2020, 2021; Obaidellah et al. 2018; Sharafi et al. 2015]. With this understanding of the experience and needs of software developers, better tools and support can be provided to enhance productivity and the quality of software.

Despite the large number of eye tracking studies in software engineering [Obaidellah et al. 2018], very few datasets are publicly available. The existence of a large dataset of eye movements in programming enables a) reproducibility of prior studies and b) accessibility of eye movement data to people who might not neceessarily have an eye tracker but are interested in analyzing/visualizing eye tracking data in unique ways. This opens the doors to whole new avenues of research. A community effort was undertaken to produce one such dataset - Eye Movements In Programming Dataset (EMIP) [Bednarik et al. 2020]. The EMIP dataset was an international and multi-institutional effort that involved eleven research teams across eight countries on four continents. The raw data of the large dataset (N=216) is freely available for download under the Creative Commons CC-BY-NC-SA license [Bednarik et al. 2020]. In order to start answering specific comprehension questions about how the gaze moves over the code, the EMIP dataset needs to be processed first.

In this paper, we present a Python library for customized post-processing of the EMIP dataset that we call the EMIP Toolkit. The toolkit is specifically designed to make the EMIP dataset easier and more accessible to directly address a researcher's comprehension questions. It implements features for fixation detection and correction, trial visualization, source code lexical data enrichment, and mapping fixation data over areas of interest in the source code. This paper makes the following contributions:

- A toolkit for customized post-processing of the EMIP dataset enabling researchers to directly query the data to answer specific comprehension questions.
- A processed version of the EMIP dataset (for all Java trials) that is filtered, corrected, and inspected with fixation data, source code token data, and lexical information tags.

Both the toolkit and the processed subset of the data are publicly available under a Creative Commons license to support future research and replication. The rest of the paper is organized to provide context to the importance and need for this toolkit and act as a description of the toolkit features and its potential use.

2 BACKGROUND AND MOTIVATION

The Eye Movements In Programming Dataset (EMIP) is the only large programming eye movements dataset that we know of that is publicly available [Bednarik et al. 2020]. EMIP was collected through an international effort that consisted of eleven research teams across eight countries on four continents. This allows for collecting data from a large number of participants (n=216) from diverse backgrounds and native languages. The same eye tracking apparatus and software were used for the data collection with consistent experimental setup.

The advantages of the EMIP dataset are a) that it is publicly available in raw unfiltered form, b) consists of a large number of participants, and c) includes data from participants with diverse levels of programming experience and native languages. These characteristics allow for new studies on themes such as the differences between eye movements of novices and experts and debugging strategies among others. At the same time, the dataset is provided in a raw data format, and the software provided by the eye tracker manufacturer is discontinued, limited in what features it provides and not free. We direct the reader to Bednarik et al. [Bednarik et al. 2020] for a complete description of the dataset.

3 EMIP TOOLKIT

In this section we provide the details of our Python library for customized post-processing of the EMIP dataset. The toolkit is specifically designed to make using the EMIP dataset easier and more accessible by providing the following functions:

- Parsing raw data files from the EMIP dataset into Experiment, Trial, and Fixation containers.
- Customizable dispersion-based fixation detection algorithm implementation according to the manual of the SMI eye tracker used in the data collection.
- Raw data and filtered data visualizations for each trial.
- Performing hit testing between fixations and AOIs to determine the fixations over each AOI.
- Customizable offset-based fixation correction implementation for each trial.
- Customizable Areas Of Interest (AOIs) mapping implementation at the line level or token level in source code for each trial.
- Visualizing AOIs before and after fixations overlay on the code stimulus.
- Mapping source code tokens to generated AOIs and eye movement data.
- Adding source code lexical category tags to eye movement data using srcML [Collard et al. 2011]. srcML is a static analysis tool and data format that provides very accurate syntactic categories (method signatures, parameters, function names, method calls, declarations and so on) for source code. We use it to enhance the eye movements dataset to enable better querying capabilities.

The complete code for the toolkit, a Jupyter Notebook examples/tutorial highlighting the main features of the toolkit, and the processed EMIP dataset are available at https://osf.io/djn9s/

3.1 Data Containers

The three main containers of the EMIP Toolkit are:

- Experiment: Represents all the trials and associated data for a single participant.
- Trial: Represents the samples and fixations in a single trial run.
- Fixation: Represents the data of a single filtered fixation consisting of multiple samples.

The experiment container implements a parser for raw eye tracking data files. The parser splits raw data into trials and adds the eye tracking samples from each trial to its container. The trial container stores data including trial number, participant ID, and the stimulus that is used in the trial. The trial container also implements a dispersion-based fixation detection algorithm that converts raw samples into fixations. In addition, the container implements fixation correction by offset, and trial data visualization. The fixation container stores the fixations generated by the detection algorithm including trial ID, participant ID, fixation timestamp, fixation duration, fixation coordinates, and the code token the fixation overlays.

The three containers are able to parse all raw data files and provide a structured view of the data that makes processing data easier. All other features of the EMIP Toolkit are implemented as free functions to make using them independent from the specific structure of the EMIP dataset.

3.2 Fixation Detection

The EMIP Toolkit implements a dispersion-based fixation detection algorithm. This algorithm distinguishes fixations form saccades, blinks, and errors based on the temporal and spacial dispersion of eye tracking samples [Nyström and Holmqvist 2010]. The main event that is often studied is fixation duration, since fixation duration is considered an indicator of information processing in human cognition [Rayner 1998]. Therefore, the most important pieces of information that a fixation detection algorithm produces are fixation location (coordinates) on the screen, and fixation duration.

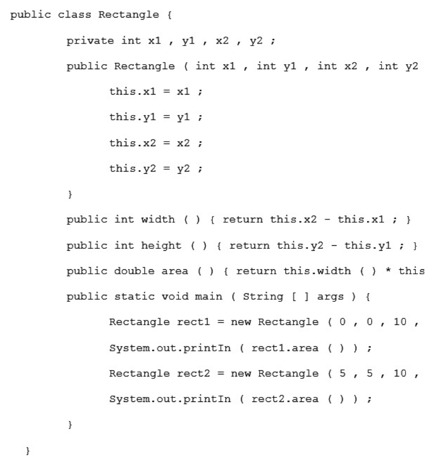
Among the many types of fixation detection algorithms, the dispersion-based algorithms are most commonly used to detect fixation events. The concept of dispersion-based algorithms consists of identifying the raw eye tracker samples as belonging to a single fixation when the samples are dispersed tightly within a limited region on the screen for a minimum period of time (in our implementation this is the parameter minimum_duration and it is set by default to 50 milliseconds). Under this type of algorithm, spatial and temporal information about samples are taken in consideration to detect fixations and saccades that are implicitly detected as based on the time and jumps between fixations [Nyström and Holmqvist 2010].

The most prominent dispersion-based fixation detection algorithm is Dispersion-Threshold Identification (I-DT) [Karthik et al. 2019; Nyström and Holmqvist 2010]. This is the preferred fixation detection algorithm by the manufacturer of the eye tracker according to their extended manual [noa 2011]. The algorithm starts with a window equal to the minimum_duration value (50 milliseconds by default), which results in excluding any fixations shorter than 50 milliseconds and considering them as noise. This window is expanded one sample at a time if the sample is within a specific dispersion radius (in our implementation it is called maximum_dispersion and it takes a value of 25 pixels by default). Once a sample is identified outside of the allowed dispersion radius, the samples in the window are considered a fixation and a new window starts. The dispersion calculation takes in account vertical and horizontal distances and it is calculated as shown in Equation 1. The potential sample is added to the window, and if the dispersion value is greater than the threshold the most recent sample is removed. Equation 1 expects $window_x$ and $window_y$ to be lists of the x, y coordinates of the samples in the window including the most recent sample that is being evaluated. The equation calculates the difference between the furthest samples in the window, and that dispersion value is then compared to the maximum_dispersion threshold. When all of the samples belonging to a fixation are detected in the window, the fixation coordinates are registered at the centroid of the window points.

$$Dispersion = (max(window_x) - min(window_x)) \\ + (max(window_y) - min(window_y)).....(1)$$

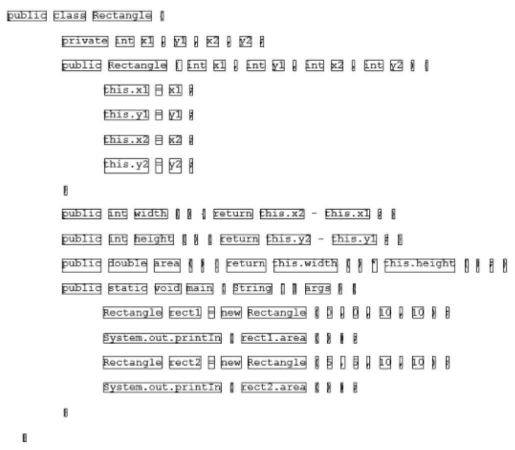
3.3 Generating Area Of Interests

An important factor in analyzing eye tracking data is to decide on the areas of interest on the stimuli to focus the analysis on. The two predominant areas in eye tracking studies with source code are the line-level and the token-level areas of interest. In line-level studies the eye tracking behaviour is studied with each line of code as a single unit, and such studies can compare the duration of time a programmer spends on a function prototype in comparison to other source code lines. Similarly, token-level studies focus on the eye movement behaviour on each code token - any set of characters that is surrounded by spaces. Token-level studies can compare eye movement behaviour on high frequency code tokens in comparison to low frequency tokens [Al Madi 2020].



```
public class Rectangle {
        private int x1 , y1 , x2 , y2 ;
        public Rectangle ( int x1 , int y1 , int x2 , int y2
              this.x1 = x1 ;
              this.y1 = y1 ;
              this.x2 = x2 ;
              this.y2 = y2 ;
        }
        public int width ( ) { return this.x2 - this.x1 ; }
        public int height ( ) { return this.y2 - this.y1 ; }
        public double area ( ) { return this.width ( ) * this
        public static void main ( String [ ] args ) {
              Rectangle rect1 = new Rectangle ( 0 , 0 , 10 ,
              System.out.printIn ( rect1.area ( ) ) ;
              Rectangle rect2 = new Rectangle ( 5 , 5 , 10 ,
              System.out.printIn ( rect2.area ( ) ) ;
        }
  }
```

(a) Stimulus before adding areas of interests.



(b) Stimulus after adding areas of interests.

Figure 1: EMIP stimulus "Java Rectangle" visualization before and after adding Areas of Interest (AOIs).

In the EMIP Toolkit we implement a function for drawing Areas of Interest (AOIs) around stimuli from the EMIP dataset on the token-level and line level. Figure 1 shows a sample AOI allocation for the stimulus "Java Rectangle" where AOIs are allocated at the token level. The function **find_rectangles** in the toolkit provides the following information for each area of interest:

- kind: Analysis level, and takes one of two values - "sub-line" (AOIs calculated at the token level) , "line" (AOIs calculated at the line level).
- name: Line and part number for each token in the code. For example, the first token in the first line would be named "line 1 part 1."

- x: X-coordinate of the upper left corner of the AOI.
- y: Y-coordinate of the upper left corner of the AOI.
- width: Width of the AOI in pixels.
- height: Height of the AOI in pixels.
- image: Name of the stimulus file used in the trial.

3.4 Trial Data Visualization

Trial data visualization is important for inspecting the validity of the data collected in each trial. EMIP Toolkit provides a single function that allows for customized visualization of any trial from any subject in the EMIP dataset. Figure 2 shows a sample visualization of trial 5 by subject 12 showing raw samples in red and detected fixation from the I-DT algorithm described earlier in green.

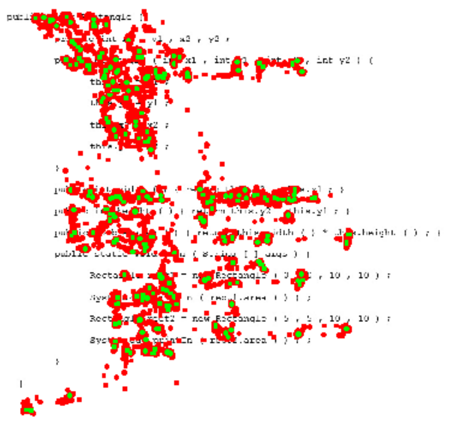


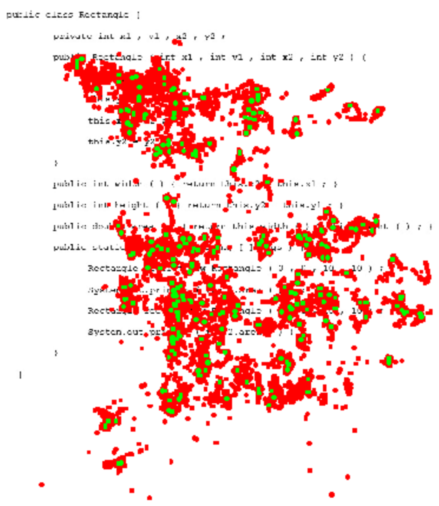
Figure 2: Sample visualization of trial 5 by subject 12 showing raw samples (in red) and detected fixation (in green).

The EMIP Toolkit function **draw_trial** generates an image of the trial visualization with the same resolution as the trial stimulus. The function takes the following parameters:

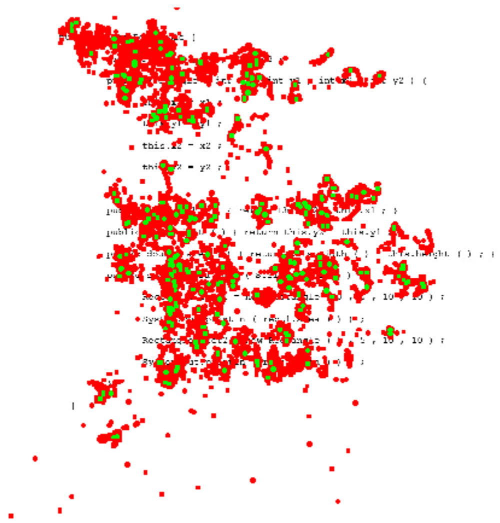
- images_path: Path for the trial stimulus image.
- draw_raw_data: Boolean indicating whether raw samples are drawn.
- draw_filtered_fixations: Boolean indicating whether detected fixations from the I-DT algorithm are drawn.
- save_image: Boolean indicating whether the resulting visualization should be saved as an image.

3.5 Fixation Correction Through Offset

There are many types of systematic errors that could affect an eye tracking study as stated in Al Madi et al. [Al Madi 2020]. This kind of error is caused by an invalidation to the calibration process that is caused by the subject moving or an inconsistency between the setup during calibration and the experiment. In many of these cases, the recorded fixations can shift from their actual position on a line of text/code and result in an erroneous recording of fixation position in a trial. Many of these erroneous trials that fall below a specific quality threshold are often filtered and neglected by researchers depending on the research goals. In some cases, applying an offset



(a) Error: example shows shifting down and to the right.



(b) Corrected: using offset (X: -100, Y: -100).

Figure 3: An example of a fixation correction by applying an offset.

to all fixations can result in restoring the erroneous fixations to their original correct position.

The EMIP Toolkit provides a fixation correction function that applies an offset to any trial that can be used with the trial visualization function by an expert to salvage erroneous eye tracking trials. The function **sample_offset** takes an x-offset and y-offset as arguments and updates the positions of all trial samples. It is designed to be called any number of times, and it keeps a history of applied offsets that is recorded with the trial visualization and can be reset (undone) at a later time.

Figure 3-a shows a trial with shifting error, where all the samples are shifted to the right and down. Considering the large number of trials in the EMIP dataset this pattern repeats in many trials.

Figure 3-b shows the corrected trial after applying an offset (x:-100, y:-100) that salvaged this trial from exclusion. In the latter section of this paper we present a filtered-corrected dataset where we apply offsets to the EMIP trials to correct them and score each trial with a numerical value representing eye tracking recording quality.

3.6 Mapping Sources Code Tokens and srcML Tags

Bounding boxes are generated for all tokens in the image stimulus. In this context, a token is classified as a collection of consecutive characters delineated by whitespace. Each bounding box is associated with a line of the source code image stimulus and the order in which the token appears on the line. The textual content of each bounding box is derived from a text version of the source code using the line and order of the provided bounding box data. This involves using whitespace on each line of the text version to split the contents into a one to one mapping with the bounding boxes. In addition to supplying the source code text contained within the bounding boxes, detailed syntactic information for each token is provided using a secondary processing phase using srcML [Collard et al. 2011].

srcML (srcML.org) is both an XML markup format for source code and an application capable of generating the aforementioned markup document. The srcML infrastructure supports the C, C++, C#, and Java programming languages. The srcML markup format uses XML to represent the hierarchy of the source code and tag information to identify the syntactic context for all textual tokens within an input source document. The srcML format ensures that all original source code content is preserved including comments and whitesapce to prevent any content loss during conversion to srcML and back to the source code format.

When converting source code to the srcML format, using the `--position` option adds attributes to each tag indicating the start and end of the line and column where each element resides within the source code file. Since the bounding box information from the EMIP image stimulus indicates which source line contains a given token, a traversal of the XML DOM can be used to find all tags that represent a line of source content. Using the subset of tags for a given line, the textual tokens contained within the XML tags can be examined to determine matches with the text version of the source code. Once a token is identified, all the tag names that encompass that token are stored to represent the complete hierarchy of syntactic context for a given source code element.

Each syntactic context collection is presented as a list of the srcML tag names separated by -> to indicate the direction of the hierarchy. This representation allows for analysis at varying levels of granularity when considering the role of syntactic elements in program comprehension and provides additional value to the existing EMIP dataset. To further simplify this process, the syntactic context provided by srcML is pre-computed and provided along with the EMIP Toolkit and dataset to minimize run-time overhead and reduce development efforts of potential users.

3.7 Mapping Fixations to Areas of Interest

Most eye tracking research revolves around the idea of measuring fixation duration over areas of interest. We have described fixation detection, correction, source code lexical data enrichment, and the visualization features of EMIP Toolkit in prior sections. In this section, we describe how fixations are mapped over areas of interest to generate the complete fixation data from the EMIP dataset. The EMIP Toolkit function hit_test takes as input the fixation data and the generated AOIs of a specific trial to calculate the fixation duration over each AOI. The function considers a fixation over an AOI if a fixation is within a 25 pixel radius of an AOI, this number is customizable in the fixation detection algorithm. Each fixation consists of raw samples within a 25 pixel dispersion radius, therefore we consider fixations 25 pixels away from an area of interest within that area of interest.

The resulting comma separated file consists of fixation data over each AOI. Each row in the resulting file corresponds to a single fixation with the following attributes on the fixation and AOI it overlays:

- trial: Trial number.
- participant: Participant number.
- code_file: Stimulus filename.
- code_language: Programming language of the trial.
- timestamp: Fixation timestamp.
- duration: Fixation duration in milliseconds.
- x_cord: X-coordinate of the fixation.
- y_cord: Y-coordinate of the fixation.
- aoi_x: X-coordinate of the area of interest under the fixation.
- aoi_y: Y-coordinate of the area of interest under the fixation.
- aoi_width: Width of the area of interest under the fixation in pixels.
- aoi_height: Height of the area of interest under the fixation in pixels.
- token: Source code token under the fixation.
- length: Length in character spaces of the source code token under the fixation.
- srcML: srcML tag of the source code token under the fixation.

4 CORRECTED DATASET

The EMIP dataset contains data for Java (207 trials), Python (5 trials) and Scala (4 trials) code. More than 95% of the trials were in Java. We construct a filtered, corrected, and scored dataset that is a subset of the EMIP dataset that focuses on the Java trials. The EMIP Toolkit fixation detection, visualization, adding tokens, adding srcML tags, and applying fixation overlay features were used on the EMIP dataset to generate a cleaned, processed version of the dataset for all the Java trials.

In some instances, the EMIP trial fixation data presented a clear shifting pattern with respect to the stimulus. Two of the authors split the dataset and applied a general offset correction to the x and y coordinates of the gaze data points with respect to the underlying stimulus in each trial. This process is repeated until the location of the data points in the visualization overlay the stimulus in a majority of locations during a visual inspection. Some trials showed an error pattern that can not be fixed with a general offset as describe by Al Madi in [Al Madi 2020], such trials were not included in the corrected dataset.

Following the offset correction, each of the authors reviewed the other authors correction and rated each data correction with a pass or fail vote. If a particular sample receives two pass votes, the data is accepted into the corrected dataset. If a sample received two fail votes, the data is deemed to be invalid and dropped from the new data set. For any sample receiving one pass and one fail vote, the authors met to review the data sample in question and discussed any issues to arrive at a consensus regarding the adjustments to the sample or its presence in the data set. The final corrected dataset is available for download in our artifact presented in Section 3.

5 CONCLUSIONS AND FUTURE WORK

We present the EMIP Toolkit, a Python library for customized post-processing of the EMIP dataset [Bednarik et al. 2020]. The toolkit is intended to make using the EMIP dataset easier and it implements many of the most common and needed features and algorithms for eye tracking in program comprehension studies. In addition, we present a corrected, filtered, and scored subset of the EMIP dataset that is generated using the EMIP Toolkit. This subset is ready for use by researchers interested in eye movements over source code and serves as an example of what the EMIP Toolkit is able to address. In the future, we aim to extend the EMIP Toolkit to other datasets and make it a generic toolkit for processing eye movements over source code. This goal includes adding more fixation detection algorithms and algorithms that are specific to source code, animated visualizations, and extending source code lexical data enrichment to languages other than Java. Finally, we hope that this work will contribute to the open data initiative set by the EMIP dataset, and inspires future research and replication in eye movements in programming.

REFERENCES

2011. iView X System Manual. https://psychologie.unibas.ch/fileadmin/user_upload/psychologie/Forschung/N-Lab/SMI_iView_X_Manual.pdf

Naser S Al Madi. 2020. *Modeling Eye Movement for the Assessment of Programming Proficiency.* Ph.D. Dissertation. Kent State University.

Nasir Ali, Zohreh Sharafl, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An empirical study on requirements traceability using eye-tracking. In *2012 28th IEEE International Conference on Software Maintenance (ICSM).* IEEE, 191–200.

Christoph Aschwanden and Martha Crosby. 2006. Code scanning patterns in program comprehension. In *Proceedings of the 39th hawaii international conference on system sciences.*

Roman Bednarik, Teresa Busjahn, Agostino Gibaldi, Alireza Ahadi, Maria Bielikova, Martha Crosby, Kai Essig, Fabian Fagerholm, Ahmad Jbara, Raymond Lister, et al. 2020. EMIP: The eye movements in programming dataset. *Science of Computer Programming* 198 (2020), 102520.

Roman Bednarik and Markku Tukiainen. 2006. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications.* ACM, 125–132.

Roman Bednarik and Markku Tukiainen. 2007. Analysing and Interpreting Quantitative Eye-Tracking Data in Studies of Programming: Phases of Debugging with Multiple Representations.. In *PPIG.* Citeseer, 13.

Dave Binkley, Marcia Davis, Dawn Lawrie, Jonathan I Maletic, Christopher Morrell, and Bonita Sharif. 2013. The impact of identifier style on effort and comprehension. *Empirical Software Engineering* 18, 2 (2013), 219–276.

Teresa Busjahn, Carsten Schulte, and Andreas Busjahn. 2011. Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research.* 1–9.

Michael L Collard, Michael J Decker, and Jonathan I Maletic. 2011. Lightweight transformation and fact extraction with the srcML toolkit. In *2011 IEEE 11th international working conference on source code analysis and manipulation.* IEEE, 173–184.

Martha E Crosby and Jan Stelovsky. 1990. How do we read algorithms? A case study. *Computer* 23, 1 (1990), 25–35.

Benoît De Smet, Lorent Lempereur, Zohreh Sharafi, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Naji Habra. 2014. Taupe: Visualizing and analyzing eye-tracking data. *Science of Computer Programming* 79 (2014), 260–278.

Hacı Ali Duru, Murat Perit Çakır, and Veysi İşler. 2013. How does software visualization contribute to software comprehension? A grounded theory approach. *International Journal of Human-Computer Interaction* 29, 11 (2013), 743–763.

Yann-Gaël Guéhéneuc. 2006. TAUPE: towards understanding program comprehension. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research.* 1–es.

Prateek Hejmady and N Hari Narayanan. 2012. Visual attention patterns during program debugging with an IDE. In *Proceedings of the Symposium on Eye Tracking Research and Applications.* 197–200.

Sebastien Jeanmart, Yann-Gael Gueheneuc, Houari Sahraoui, and Naji Habra. 2009. Impact of the visitor pattern on program comprehension and maintenance. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement.* IEEE, 69–78.

G Karthik, J Amudha, and C Jyotsna. 2019. A Custom Implementation of the Velocity Threshold Algorithm for Fixation Identification. In *2019 International Conference on Smart Systems and Inventive Technology (ICSSIT).* IEEE, 488–492.

Meng-Lung Lai, Meng-Jung Tsai, Fang-Ying Yang, Chung-Yuan Hsu, Tzu-Chien Liu, Silvia Wen-Yu Lee, Min-Hsien Lee, Guo-Li Chiou, Jyh-Chong Liang, and Chin-Chung Tsai. 2013. A review of using eye-tracking technology in exploring learning from 2000 to 2012. *Educational research review* 10 (2013), 90–115.

Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 1–37.

Naser Al Madi, Cole S Peterson, Bonita Sharif, and Jonathan Maletic. 2020. Can the ez reader model predict eye movements over code? towards a model of eye movements over source code. In *ACM Symposium on Eye Tracking Research and Applications.* 1–4.

Naser Al Madi, Cole S Peterson, Bonita Sharif, and Jonathan Maletic. 2021. From Novice to Expert: Analysis of Token Level Effects in a Longitudinal Eye Tracking Study. In *29th IEEE/ACM International Conference on Program Comprehension.*

Marcus Nyström and Kenneth Holmqvist. 2010. An adaptive algorithm for fixation, saccade, and glissade detection in eyetracking data. *Behavior research methods* 42, 1 (2010), 188–204.

Unaizah Obaidellah, Mohammed Al Haek, and Peter C-H Cheng. 2018. A survey on the usage of eye-tracking in computer programming. *ACM Computing Surveys (CSUR)* 51, 1 (2018), 5.

Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. 2010. An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering* 15, 5 (2010), 493–522.

Keith Rayner. 1998. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin* 124, 3 (1998), 372.

Pablo Romero, Richard Cox, Benedict du Boulay, and Rudi Lutz. 2002. Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. In *International Conference on Theory and Application of Diagrams.* Springer, 221–235.

Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering* 25, 5 (2020), 3128–3174.

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* 67 (2015), 79–107.

Bonita Sharif and Jonathan I Maletic. 2010. An eye tracking study on camelcase and under_score identifier styles. In *2010 IEEE 18th International Conference on Program Comprehension.* IEEE, 196–205.

Bonita Sharif, John Meinken, Timothy Shaffer, and Huzefa Kagdi. 2017. Eye movements in software traceability link recovery. *Empirical Software Engineering* 22, 3 (2017), 1063–1102.

Kshitij Sharma, Daniela Caballero, Himanshu Verma, Patrick Jermann, and Pierre Dillenbourg. 2015. Looking AT versus looking THROUGH: A dual eye-tracking study in MOOC context. International Society of the Learning Sciences, Inc.[ISLS].

Zéphyrin Soh, Zohreh Sharafi, Bertrand Van den Plas, Gerardo Cepeda Porras, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. Professional status and expertise for UML class diagram comprehension: An empirical study. In *2012 20th IEEE International Conference on Program Comprehension (ICPC).* IEEE, 163–172.

Randy Stein and Susan E Brennan. 2004. Another person's eye gaze as a cue in solving programming problems. In *Proceedings of the 6th international conference on Multimodal interfaces.* 9–15.

Rachel Turner, Michael Falcone, Bonita Sharif, and Alina Lazar. 2014. An eye-tracking study assessing the comprehension of C++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications.* 231–234.

Braden Walters, Timothy Shaffer, Bonita Sharif, and Huzefa Kagdi. 2014. Capturing software traceability links from developers' eye gazes. In *Proceedings of the 22nd International Conference on Program Comprehension.* 201–204.

Shehnaaz Yusuf, Huzefa Kagdi, and Jonathan I Maletic. 2007. Assessing the comprehension of UML class diagrams via eye tracking. In *15th IEEE International Conference on Program Comprehension (ICPC'07).* IEEE, 113–122.