# EMIP: The Eye Movements in Programming Dataset

Roman Bednarik[1]

*University of Eastern Finland*

Teresa Busjahn

*HTW Berlin, Germany*

Agostino Gibaldi

*University of Genova, Italy*

Alireza Ahadi

*University of Technology, Sydney, Australia*

Maria Bielikova

*Kempelen Institute of Intelligent Technologies, Slovakia*

Martha Crosby

*University of Hawaii at Mnoa, USA*

Kai Essig

*Bielefeld University, Germany*

Fabian Fagerholm

*Aalto University, Finland and University of Helsinki, Finland*

Ahmad Jbara

*Augusta University, GA, USA*

Raymond Lister

*University of Technology, Sydney, Australia*

Pavel Orlov

*Imperial College London, UK*

[1]Corresponding author

James Paterson

*Glasgow Caledonian University, UK*

Bonita Sharif

*University of Nebraska, Lincoln, USA*

Teemu Sirkiä

*Aalto University, Finland*

Jan Stelovsky

*University of Hawaii at Mnoa, USA*

Jozef Tvarozek

*Slovak University of Technology in Bratislava, Slovakia*

Hana Vrzakova

*University of Colorado, Boulder, USA*

Ian van der Linde

*Anglia Ruskin University, Cambridge, UK*

**Abstract**

A large dataset that contains the eye movements of N=216 programmers of different experience levels captured during two code comprehension tasks is presented. Data are grouped in terms of programming expertise (from none to high) and other demographic descriptors. Data were collected through an international collaborative effort that involved eleven research teams across eight countries on four continents. The same eye tracking apparatus and software was used for the data collection. The Eye Movements in Programming (EMIP) dataset is freely available for download. The varied metadata in the EMIP dataset provides fertile ground for the analysis of gaze behavior and may be used to make novel insights about code comprehension.

*Keywords:* eye-tracking, program comprehension, dataset

## 1. Introduction

The earliest studies that examined of the role of visual attention in programming date back to 1990. Crosby and Stelovsky [1] asked N=19 participants, divided into low and high experience groups, to view prose, code, and graphical versions of a binary search algorithm while their eye movements were recorded. Results included that a range of individual strategies/scanpaths were were found; that there were significant differences in the way programmers read source code in comparison to prose (e.g., that more fixations were directed to relevant areas of code in comparison to prose); that programmers with less experience spend more time examining code comments; and those with more experience examine code more efficiently, directing their attention to the most important (complex) areas of the algorithm.

With the increasing availability and maturity of eye-tracking apparatus, more studies of program comprehension using eye tracking have emerged. A number of exemplar studies highlighting the kinds of research questions that can be addressed by analysing the eye movements of programmers are briefly summarized below, but for more complete reviews see [2] and [3]. In 2006, Uwano et al. [4] presented typical patterns of eye movements across source code. Bednarik and Tukiainen [5] reported on the differences in gaze patterns between novice and expert programmers using an interactive dynamic visualization environment. More recent studies examined the effect (on the pattern of eye movements elicited) of identifier naming conventions [6, 7], programming language [8], and also examined the potential role of parafoveal vision (i.e., outside the visual axis) in code comprehension [9]. Busjahn et al. showed that the order in which novice and expert programmers read through the lines of code in a program differs from the order that those lines would be executed [10].

In the present article, we present the EMIP (Eye Movements In Programming) dataset, a large eye movement dataset recorded from programmers across multiple sites of different levels of expertise as they examined two object oriented source code fragments. It is hoped that this dataset will enable more questions concerning program comprehension to be addressed, and that the size of the dataset will allow this to be done with ample statistical power (*cf.* existing studies that typically use much fewer participants). For a practical guide on how to design and conduct eye tracking studies in software engineering we direct the reader to [11].

## 2. Motivation for Eye Movements in Programming Dataset

With the increasing number of published studies examining eye movements in programming, there is a growing need to compare and consolidate theories and results. Aside from systematic reviews [2, 3], one way to accomplish this is through the provision of a large, publicly available dataset that can be mined both to verify existing theories and develop new ones. Some of the principal motivations for the new dataset are enumerated below.

First, the question of how to exploit eye-tracking data effectively during live programming is unresolved; for instance, in the development of automated tools for error correction [12]. Such methods would greatly benefit from a large pool of data collected in controlled conditions. A similar argument holds for research using machine learning and data-mining. The training, optimization, and validation of such systems would benefit greatly from the availability of a sufficiently large quantity of labeled data.

Second, such a dataset has the capacity to inform the use of eye tracking in the programming and software development process. For example, in recent studies, eye-tracking has been used to improve awareness and collaboration between pair programmers [13]. Learning the typical gaze patterns of programmers during comprehension activities is more robust in the presence of a sufficiently large dataset.

Third, central questions in eye-tracking programming research focus on differences that emerge as a consequence of programmer expertise. Indeed, researchers have shown great interest in trying to identify and understand the diagnostic markers of expertise. A large dataset, as presented here, supported by a large number of participants of different expertise levels, allows for finer-grained analyses of expertise-related research questions.

Fourth, eye-tracking data is gaining popularity as a physiological measure of developers' workload or emotional state [14]. These studies benefit from the availability of a large dataset, providing high statistical power. Moreover, recent years have seen the development of low-cost eye tracking devices with performance that is beginning to approach research grade devices [15, 16], and the integration of eye-tracking devices into conventional laptop computers, allowing for more widespread use of these approaches in the future.

Fifth, obtaining a large dataset requires significant technical investment, effort, and is costly to collect. A large, free dataset should help support the enlargement of the research community in this area, permitting both the replication and validation of existing findings and the development of new

avenues of research in the sub-field of program comprehension in software engineering.

This paper describes an international effort to collect a large and carefully controlled dataset that is suitable for addressing the questions and research problems described above, *inter alia*.

## 3. Materials and Methods

We describe the logistics of the data collection process, test stimuli (i.e., the code that participants were asked to examine), apparatus, the experimental procedure, and the format and structure of the captured data in detail below. This information is provided to enable users to evaluate the robustness of our data, to understand the kinds of research questions that can be asked (i.e., which variables describing participants were collected and therefore may serve as predictors in analyses), to enable others to replicate and/or extend the dataset, and to enable others to compare our results with their own by considering any methodological differences.

To support replication, all materials for conducting the study are available at `http://emipws.org/stimulus-material/`.

### *3.1. Data collection logistics*

The EMIP dataset was collected as a community effort involving eleven research teams across eight countries and four continents. A call for participation was distributed using mailing lists likely to be used by those with an interest in the topic of eye movements in programming. SensoMotoric Instruments (SMI) kindly provided two eye-movement recording systems (comprising a laptop computer, software, and eye tracking hardware, described in detail in Section 3.2, below) that were shipped to participating labs, along with detailed instructions on how to assemble the hardware and how to run the experimental software. This high resolution eye-tracking system was portable, enabling it to be posted to data collection sites, and the availability of two systems enabled labs to work concurrently, thereby speeding up data collection.

Assistance was provided via email, when needed. Data were collected at the following sites:

- The Centre for Human Centred Technology Design, University of Technology Sydney, Australia;

- The Department of Computer Science, Aalto University, Finland;
- The Department of Computer Science, University of Helsinki, Finland;
- The Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, Slovakia;
- Information & Computer Sciences, University of Hawaii at Mānoa, USA;
- Neuroinformatics Group, Bielefeld University, Germany;
- The School of Mathematics and Computer Science of the Netanya Academic College, Netanya, Israel;
- The School of Computing, Engineering and Built Environment, Glasgow Caledonian University, United Kingdom;
- Software Engineering Research and Empirical Studies Lab, Youngstown State University, USA;
- The Physical Structure of Perception and Computation Group, University of Genoa, Italy;
- The School of Computing and Information Science, Anglia Ruskin University, Cambridge, United Kingdom.

### *3.2. Apparatus*

Eye movements were recorded using a non-invasive screen-mounted SMI RED250 mobile video-based eye tracker. The eye tracker provided has a sample rate of 250Hz, with an accuracy of $< 0.4^\circ$ and a precision of $\approx 0.03^\circ$ of visual angle. The working distance from the device is $50 - 80$ cm within a 'head box' of $32 \times 21$ cm at 60 cm, which provided an ideal workspace for the experimental procedure (see Section 3.4).

Stimuli were presented on a laptop computer screen set at a resolution of $1920 \times 1080$ pixels. Stimuli were free-viewed (i.e., no head or chin rest was used) to simulate a naturalistic programming environment (something that would not have been possible had a head-mounted eye tracker or head/chin restraint been used).

The data collection procedure (see below) was implemented in the SMI Experimental Suite (a software bundle that was packaged on the laptop).

The experimental apparatus, setup and software were matched as closely as possible between collaborating sites by shipping a pre-configured eye tracker and laptop computer. Data were collected in a quiet, well-lit environment to minimize distractions to participants.

### *3.3. Participants*

Participants were recruited at each site by opportunity sampling. Data from N=216 participants are included in the dataset, of whom 41 were female and 175 were male (mean age 26.56 years, SD = 9.28). All participants completed a demographic questionnaire, summarized in Table 1. Participants were principally University students enrolled in undergraduate or postgraduate courses related to computing, but also included academic and administrative staff and some professional programmers.

Participants came from a diverse pool of language families (1 Arabic, 2 Bengali, 1 Cantonese, 4 Chinese, 2 Czech, 1 Egyptian, 62 English, 1 English and Hebrew, 17 Finnish, 10 German, 2 Greek, 8 Hebrew, 3 Hindi, 21 Italian, 1 Italian and English, 1 Marathi, 2 Nepali, 1 Norwegian, 1 Persian, 2 Portuguese, 1 Punjabi, 1 Russian and Hebrew, 57 Slovak, 3 Spanish, 2 Swedish, 1 Tagalog, 1 Tamil, 4 Telugu, 1 Thai, 1 Turkish, 1 Ukrainian). Out of 154 non-native speakers, 66 participants spoke English fluently. 84 participants reported medium English proficiency and 4 participants reported low English proficiency.

All participants had normal or corrected-to-normal vision (17 were wearing contact lenses, 74 glasses). Ethics clearance for the study was granted at all sites. Participation was voluntary, and participants were treated in accordance with the tenets of the Declaration of Helsinki. No payment was offered.

### *3.4. Experimental procedure*

Participants were seated in front of the laptop that had the eye tracker installed on it. When participants indicated that they were ready to proceed, an instruction screen was presented explaining what they were being asked to do. Next, a questionnaire was presented. This included identifying the programming language that they wished to be used in the experiment (i.e., the language that they were most familiar with). Three language options were provided: Java, Scala, or Python. Programming expertise was self-evaluated as none, low, medium or high, and number of years of programming experience was also recorded.

Table 1: Metadata provided in emip_metadata.csv (as part of the dataset).

| **Variable** | **Description** | **Value** |
|---|---|---|
| id | Unique identifier, which refers to the raw gaze data file | [n] |
| age | Age | [years] |
| gender | Gender | [male, female, other] |
| mother_tongue | Mother tongue | [full-text] |
| English_level | English proficiency | [low, medium, high] |
| visual_aid | Is the participant wearing glasses or contact lenses | [no, glasses, contact lenses] |
| makeup | Is the participant wearing mascara or other eye-make-up | [yes, no] |
| experiment_language | Programming language used in the experiment | [Java, Python, Scala] |
| expertise_experiment_language | Expertise in Java/Python/Scala | [none, low, medium, high] |
| time_experiment_language | How long the participant has been programming in Java/Python/Scala | [years] |
| frequency_experiment_language | How often does the participant program in Java/Python/Scala | [not at all, less than 1h/m, less than 1h/w, less than 1h/d, more than 1h/d] |
| other_languages | Other programming languages the participant knows | [language_level of expertise] |
| expertise_programming | Overall programming expertise | [none, low, medium, high] |
| time_programming | How long the participant has been programming | [years] |
| frequency_other_language | How often the participant uses programming languages other than Java/Python/Scala | [not at all, less than 1h/m, less than 1h/w, less than 1h/d, more than 1h/d] |
| For each stimulus program: | | |
| answer_{rectangle—vehicle} | Answer to the comprehension question | [full-text] |
| correct_{rectangle—vehicle} | Evaluation of the answer | [0,1] |
| order_{rectangle—vehicle} | Order in which the stimulus programs were shown | [1,2] |
| stimulus_{rectangle—vehicle} | Filename of the screenshot in folder "stimuli" | [full-text] |
| {mother_tongue—time_experiment_language—time_programming—other_languages}_original | unedited participant entries | [full-text] |

Next, the eye tracker was calibrated using a 9-point calibration routine, and its accuracy checked with a validation procedure. This required participants to attend predefined regions of interest (ROIs) while the experimenter visually checked that gaze and the regions coincided correctly.

Following successful calibration, participants completed two code comprehension tasks (Vehicle and Rectangle, each comprising 11-22 lines of code), presented in the same order for all participants. Participants were instructed to read and try to understand the code, and to press space bar when they were done. Next, a multiple-choice question was presented on the screen that evaluated code comprehension. No time limit to answer the question was applied. At the end of the experiment, eye movement coordinates and question responses were stored for offline analysis.

### *3.5. Code and comprehension questions*

The code presented to participants was chosen to be simple enough to be understood by novices, yet not too trivial for experts. In particular, static metrics such as Cyclomatic Complexity [17] and control structure nesting indicate that the code was simple, whereas the results of the comprehension questions (See Section 4) show that they were not necessarily too trivial for the participants. If more complex code had been used then we may have risked inexperienced programmers giving up or examining the code pseudo-randomly. Furthermore, the code was short enough to fit onto a single screen without scrolling, enabling straightforward eye movement analysis.

*Rectangle:*

The Rectangle code defines a class Rectangle that contains four coordinate variables, a constructor, and methods to compute area, width, and height. In the *main* method, two rectangle objects are instantiated and their areas calculated. It was adapted from a code comprehension study written in Python [18] which we translated to Java and Scala. The comprehension question for the Rectangle task is shown in Table 2.

*Vehicle:*

The Vehicle code defines a class Vehicle that contains a number of variables, a constructor, and an accelerate method that could modify a current speed variable. In a *main* method, a single object is instantiated and its speed subsequently modified. The comprehension question for the Vehicle task is shown in Table 3.

Table 2: Multiple choice comprehension question for the Rectangle code

| |
|---|
| *The program:*<br>• *computes the area of rectangles by multiplying their width (x1-x2) and height (y1-y2).*<br>• *computes the area of rectangles by multiplying their width (x2-x1) and height (y2-y1).*<br>• *computes the area of rectangles by multiplying their width (x1-y1) and height (x2-y2).*<br>• *I'm not sure.* |

### *3.6. Dataset structure and contents*

The dataset is available for download as a 560MB ZIP file at `http://emipws.org/wp-content/uploads/emip_dataset.zip`. It is distributed under the Creative Commons CC-BY-NC-SA license. Table 4 lists the contents of the package. The eye movement data is in a generic *.tsv* (tab separated value) format to maximize compatibility with analysis software.

In order to allow for automatic processing, some of the information provided by the participants required editing: (1) multiple answers were separated by a semicolon (e.g., two or more native languages were provided); (2) text in answers to numeric questions was converted to numbers (e.g., *one year* was converted to *1*); (3) redundant information was removed. The exact information entered by the participants is also retained, in the columns with the same name and "_original" added (see Table **??**).

## 4. Results

This section provides the accuracy results for each comprehension question along with some descriptive statistics on programming languages used and participant expertise.

### *4.1. Code comprehension results*

Table 5 summarizes the number of correct and incorrect answers for both items of code examined. Most participants responded correctly to the ques-

Table 3: Multiple choice comprehension question for the Vehicle class

*The program:*

- *defines a vehicle by producer that has a type and can reduce its speed.*
- *defines a vehicle by producer that has a type and can accelerate its speed.*
- *defines a vehicle by producer that has a type and can accelerate and reduce its speed.*
- *I'm not sure.*

Table 4: Overview of dataset content

| **Content** | **Description** | **Size** |
|---|---|---|
| rawdata | folder with 216 TSV-files containing raw eye movement data | 2.5 GB |
| stimuli | folder with screenshots of the experiment slides in JPG-format and CSV-files with AOI coordinates for the stimulus programs | 1 MB |
| emip_metadata | CSV file with participants' background information, order in which the stimulus programs were show and information about the comprehension questions | 93 kB |
| date | TXT-file specifying when the dataset was uploaded | 13 B |

tion about the *Rectangle* code, but fewer did so for the *Vehicle* code. The majority of participants understood the general idea of the Vehicle program, but did not realise that the (signed) datatype used as an argument to the method that modified the value of the speed variable supported the possibility of decreasing as well as increasing the speed of vehicle objects (i.e.,

| Task | Correct | Incorrect | Total |
|---|---|---|---|
| Rectangle | 152 | 64 | 216 |
| Vehicle | 50 | 166 | 216 |
| Total | 202 | 230 | 432 |

Table 5: Crosstabulation of task performance.

that passing a negative integer to the accelerate method would decrease the speed of the vehicle). Hence, even though it is not a complex program, many participants did not fully grasp this more subtle nuance of the language.

Whilst negative acceleration, in physics, can decrease speed, one might argue that our name for the accelerate method was misleading in relation to the question posed given the expertise of the target audience (i.e., the question asked whether speed reduction was possible, and in the vernacular the term accelerate is commonly taken to mean increase speed), which will have increased the number of incorrect responses, despite being technically valid. It is important to note that participants did not know what they would be asked after they had examined the code, so this should not have affected the distribution of eye movements, as participants were instructed to examine the code in order to understand it. Figure 3 represents the fixation density map for one participant for both code stimuli. The fixation density map was computed using EMA, a free Eye Movement Analysis toolbox [19].

### *4.2. Programming languages*

Most participants elected to have the code presented in Java (95.83%), potentially reflecting the continued widespread use of Java in undergraduate teaching and in industry. A much smaller number of participants selected Python (2.31%) or Scala (1.85%). In the questionnaire, participants reported having expertise in a wide variety of other languages (see Figure 2). Interestingly, C together with its extensions and derivatives (Handel-C, Embedded C, C++, C#, Objective-C) was the language mentioned most often (81%), followed by Python (31%), and JavaScript (26%).

### *4.3. Participant expertise*

As noted above, participants indicated their level of expertise in the programming language used in the experiment. The distribution of expe-

rience levels for our participants was: none (13.89%), low (31.94%), medium (46.29%), and high (7.87%). On average, participants has 2.29 years (SD = 3.34) of experience in the programming language selected for the experiment.

This information can be used to examine correlations in the eye tracking data to participant expertise. For example, in Figure 4 low expertise (*e.g.*, null or small) is characterized by gaze density maps with greater spatial dispersion across the code page, potentially indicating a more exploratory approach rather than one that is focused on the most important/diagnostic features of the program. Similarly, Figure 5 shows how participants with low expertise produced more spatially distributed fixations, and fixations of longer duration, compared to expert participants. Note that these are cursory high-level observations and more detailed analysis is needed to learn more about how expertise affected the results.

## 5. Discussion

Experimenter and participant time, equipment cost and availability, the provisioning and maintenance of repositories, data processing skills, and other factors limit the availability of large datasets of eye movements. By distributing the efforts across a number of sites, we reduced some of these costs in the creation of this EMIP dataset. In addition, the collaborative knowledge, skills, peer-support and discussion allowed us to support the validity of the setup and the resulting data.

The EMIP dataset presents a range of possible use cases, some of which were outlined above. Relating gaze behavior with participants' programming expertise and other metadata can potential reveal novel insights concerning the relationship between code comprehension and demographic variables.

Low-level eye movement parameters observed in reading text, from [20], are listed below:

- *Saccade frequency* - Experienced readers make a saccade during reading every quarter of a second on average.
- *Fixation duration* - The average fixation duration is 200-250ms, and the range is 100ms to over 500ms.
- *Saccade amplitude* - At each saccade, the eyes move forward a number of characters that varies from 1 to 20, with the average being 7-9 characters.

- *Saccade duration* - Saccades are relatively short and on average last for 20-40 ms.

The large size of the dataset can provide baseline data that highlights how reading source code may differ from reading of text. For instance, source code may elicit different kinds of low-level eye movement parameters compared to examining images [21][22] or reading prose [20], given that since code is not typically read sequentially and will likely entail repeated regressions to particularly important areas. In addition to the metrics listed in [20], we direct the reader to [23] for a list of eye movement metrics used in software engineering studies.

Along with the programming language experience and other metadata, our dataset could be used in predictive models of expertise by examining the efficiency of the code examination process. This has potential applications in teaching, assessment and recruitment (although clearly such data must be treated cautiously). To accomplish this, deep learning networks trained on expertise-labeled eye movement data could be used [24].

Other potential uses of the dataset unrelated to program comprehension research include: (i) to evaluate the potential of eye-movement-based biometric identification systems, in which the oculomotor behavior of an individual potentially represents a uniquely identifiable signature [25]; (ii) to evaluate the degree to which participants calibration is aligned correctly with expected regions of interest (here, lines of text in a computer program), enabling eye tracker accuracy and precision to be evaluated; (iii) to compare the eye movement data with that obtained using consumer-grade web-cam based eye trackers, which are just beginning to offer reasonable levels of accuracy (e.g., [26][27]).

## 6. Limitations

The present study has a number of limitations worth highlighting: (i) Only two code fragments were examined by participants, and both were object oriented, thus any findings may or may not generalise to more algorithmic code or code written in languages in other programming paradigms; (ii) Since this was a multi-site study, small differences in experimental setup may have occurred, despite the same eye tracker and laptop computer being shipped to all sites to try to standardize to the greatest degree possible; (iii) The code comprehension questions used, although administered *post-hoc* (i.e.,

did not affect the eye movements elicited during code examination) were, in retrospect, quite limited. The first question could have been answered using algebraic knowledge, and the second may have been affected by some participants not knowing that negative acceleration is standard terminology in physics to elicit a reduction in velocity, and thus that the accelerate method could validly accept a negative argument.

## 7. Conclusions and Future Work

In this article, a large dataset that contains the eye movements of programmers recorded during two code comprehension tasks is presented. The data were collected collaboratively across eleven research teams, and were subsequently organized and cleaned, and published in a public (online) repository that can be found at (`http://emipws.org`). Extensive metadata is provided that can be used to address a wide variety of research questions. The dataset is sufficiently large and varied to enable code comprehension questions to be addressed with ample statistical power.

Given the limitations outlined in the previous section, future work could usefully be directed to collect the eye movement of programmers while examining code written in languages that use other programming paradigms (i.e., not just object oriented), code spanning a broader range of difficulty levels (e.g., algorithms of greater complexity), and for which a greater number and variety of comprehension questions were asked. In addition, we welcome the program comprehension and eye tracking community to use the dataset and extend it with other post processing and analyses.

## Acknowledgements

The authors would like to thank all participants who took part in the study.

## 8. References

[1] M. E. Crosby, J. Stelovsky, How do we read algorithms? A case study, Computer 23 (1) (1990) 25–35.

[2] A. H. M. . C. P. C. H. Obaidellah, U., A survey on the usage of eye-tracking in computer programming, ACM Computing Surveys 51 (1) (2018) 1–58.

[3] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, A systematic literature review on the usage of eye-tracking in software engineering, Information and Software Technology 67 (2015) 79–107.

[4] H. Uwano, M. Nakamura, A. Monden, K.-i. Matsumoto, Analyzing individual performance of source code review using reviewers' eye movement, in: Proceedings of the 2006 symposium on Eye tracking research & applications, ACM, 133–140, 2006.

[5] R. Bednarik, M. Tukiainen, An eye-tracking methodology for characterizing program comprehension processes, in: Proceedings of the 2006 symposium on Eye tracking research & applications, ACM, 125–132, 2006.

[6] B. Sharif, J. I. Maletic, An eye tracking study on camelcase and under_score identifier styles, in: Program Comprehension (ICPC), 2010 IEEE 18th International Conference on, IEEE, 196–205, 2010.

[7] D. W. Binkley, M. Davis, D. J. Lawrie, J. I. Maletic, C. Morrell, B. Sharif, The impact of identifier style on effort and comprehension, Empirical Software Engineering 18 (2) (2013) 219–276, URL `https://doi.org/10.1007/s10664-012-9201-4`.

[8] R. Turner, M. Falcone, B. Sharif, A. Lazar, An eye-tracking study assessing the comprehension of C++ and Python source code, in: Proceedings of the Symposium on Eye Tracking Research and Applications (ETRA), ACM, 231–234, 2014.

[9] P. A. Orlov, R. Bednarik, The role of extrafoveal vision in source code comprehension, Perception 46 (5) (2017) 541–565.

[10] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, S. Tamm, Eye movements in code reading: Relaxing the linear order, in: Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on, IEEE, 255–265, 2015.

[11] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, M. Crosby, A Practical Guide on Conducting Eye Tracking Studies in Software Engineering, Empirical Software Engineering (2020) in press.

[12] C. Palmer, B. Sharif, Towards automating fixation correction for source code, in: Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, ACM, 65–68, 2016.

[13] S. D'Angelo, A. Begel, Improving Communication Between Pair Programmers Using Shared Gaze Awareness, in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, ACM, 6245–6290, 2017.

[14] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, M. Züger, Using psycho-physiological measures to assess task difficulty in software development, in: Proceedings of the 36th International Conference on Software Engineering, ACM, 402–413, 2014.

[15] K. Ooms, L. Dupont, L. Lapon, S. Popelka, Accuracy and precision of fixation locations recorded with the low-cost Eye Tribe tracker in different experimental setups, Journal of Eye Movement Research 8 (1).

[16] A. Gibaldi, M. Vanegas, P. J. Bex, G. Maiello, Evaluation of the Tobii EyeX Eye tracking controller and MATLAB toolkit for research, Behavior Research Methods 49 (3) (2017) 923–946.

[17] T. J. McCabe, A Complexity Measure., IEEE Trans. Software Eng. 2 (4) (1976) 308–320, URL `http://dblp.uni-trier.de/db/journals/tse/tse2.html#McCabe76`.

[18] M. Hansen, Quantifying code complexity and comprehension, Ph.D. thesis, Indiana University, 2015.

[19] A. Gibaldi, S. Sabatini, The saccade main sequence revised: a fast and repeatable tool for oculomotor analysis, Behavior Research Methods .

[20] K. Rayner, B. J. Juhasz, A. Pollatsek, Eye movements during reading, The Science of Reading: A Handbook (2005) 79–97.

[21] I. van der Linde, U. Rajashekar, A. C. Bovik, L. K. Cormack, DOVES: A database of visual eye movements, Spatial Vision 22 (2) (2009) 161–177.

[22] U. Rajashekar, I. van der Linde, A. C. Bovik, L. K. Cormack, Foveated analysis and selection of visual fixations in natural scenes, in: Proc. IEEE Int. Conf. Image Processing (ICIP), Atlanta GA, IEEE, 453–456, 2006.

[23] Z. Sharafi, T. Shaffer, B. Sharif, Y. Guéhéneuc, Eye-Tracking Metrics in Software Engineering, in: J. Sun, Y. R. Reddy, A. Bahulkar, A. Pasala (Eds.), 2015 Asia-Pacific Software Engineering Conference, APSEC 2015, New Delhi, India, December 1-4, 2015, IEEE Computer Society, 96–103, doi:\let\@tempa\bibinfo@X@doi10.1109/APSEC.2015.53, URL https://doi.org/10.1109/APSEC.2015.53, 2015.

[24] M. Kümmerer, T. S. Wallis, M. Bethge, DeepGaze II: Reading fixations from deep features trained on object recognition, arXiv preprint arXiv:1610.01563 .

[25] T. Busjahn, C. Schulte, B. Sharif, A. Begel, M. Hansen, R. Bednarik, P. Orlov, P. Ihantola, G. Shchekotova, M. Antropova, et al., Eye tracking in computing education, in: Proceedings of the tenth annual conference on International computing education research, ACM, 3–10, 2014.

[26] A. Canessa, A. Gibaldi, M. Chessa, S. P. Sabatini, F. Solari, The perspective geometry of the eye: toward image-based eye-tracking, in: Human-Centric Machine Vision, IntechOpen, 2012.

[27] A.-H. Javadi, Z. Hakimi, M. Barati, V. Walsh, L. Tcheang, SET: A pupil detection method using sinusoidal approximation, Frontiers in Neuroengineering 8 (2015) 4.

```
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.height ( ) ; }
       public static void main ( String [ ] args ) {
            Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 ) ;
            System.out.printIn ( rect1.area ( ) ) ;
            Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 ) ;
            System.out.printIn ( rect2.area ( ) ) ;
       }
  }
```

(a) Rectangle

```
public class Vehicle {
       String producer , type ;
       int topSpeed , currentSpeed ;

       public Vehicle ( String p , String t , int tp ) {
              this.producer = p ;
              this.type = t ;
              this.topSpeed = tp ;
              this.currentSpeed = 0 ;
       }

       public int accelerate ( int kmh ) {
              if ( ( this.currentSpeed + kmh ) > this.topSpeed ) {
                     this.currentSpeed = this.topSpeed ;
              } else {
                     this.currentSpeed = this.currentSpeed + kmh ;
              }
              return this.currentSpeed ;
       }

       public static void main ( String args [ ] ) {
              Vehicle v = new Vehicle ( "Audi" , "A6" , 200 ) ;
              v.accelerate ( 10 ) ;
       }
}
```

(b) Vehicle



Figure 1: Code in Java

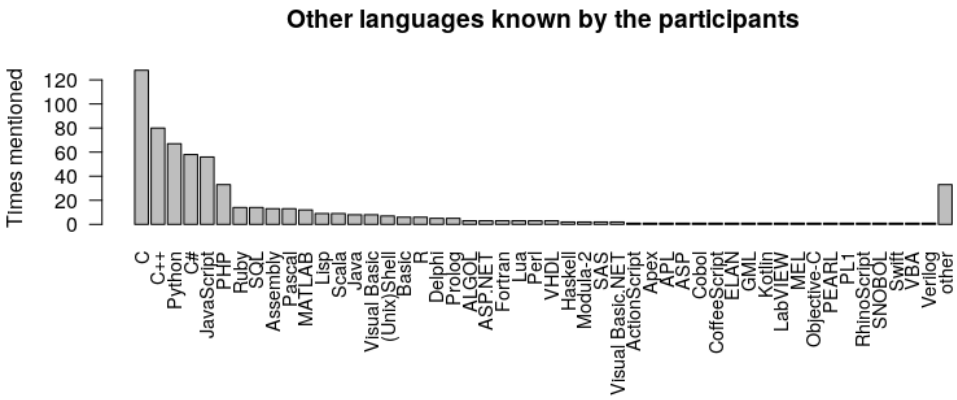


Figure 2: Programming languages that the participants claimed to know in addition to the language used in the experiment. C includes C-based derivatives such as Handel-C and Embedded C. The category *other* includes all entries that are not programming languages strictly speaking (including Arduino, Closure, CSS, Excel, HTML, Unix, and XML).

RECTANGLE


VEHICLE


Figure 3: **Gaze density maps of a single participant for Rectangle (top) and Vehicle (bottom) code.** Computed using a Gaussian kernel density function wherein red denotes a high density of fixations and green a low density.

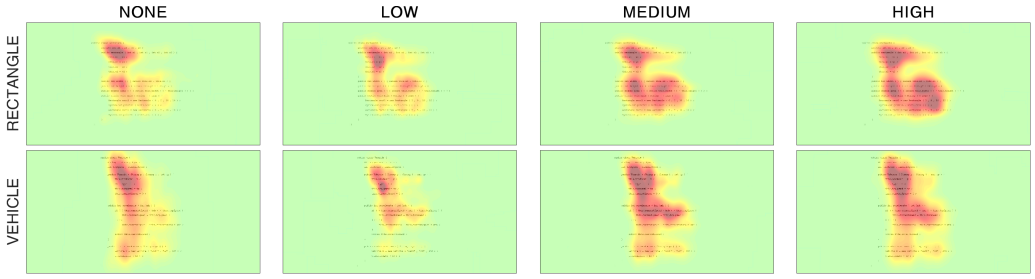


Figure 4: **Gaze density maps grouped by programming expertise for Rectangle (top) and Vehicle (bottom) code.** The maps are computed by grouping the participants into expertise levels, from left to right *none*, *low*, *medium* or *high*. Computed using a Gaussian kernel density function wherein red denotes a high density of fixations and green a low density. Each map represents the mean among of the fixation density maps across participants in each group.

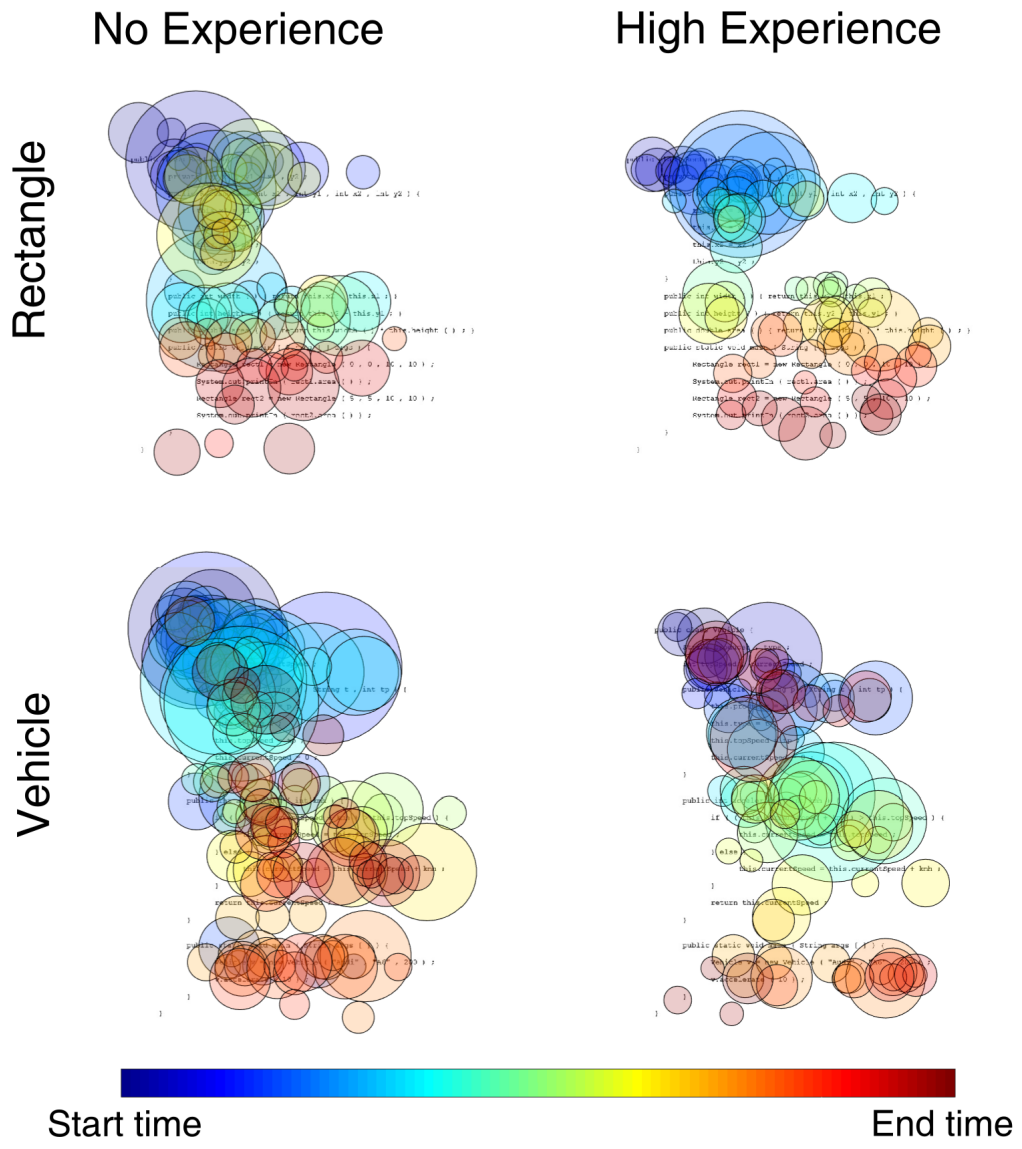


Figure 5: **Distribution of fixations for expert and inexpert participants.** Fixation patterns corresponding to participants with no (left) and high expertise (right), for the Rectangle (top) and Vehicle (bottom) code. Each circle represents a fixation, and the radius is proportional to the fixation duration. Blue colors correspond to the start of the trial while red colors to the end of the trial, according to the colorbar at the bottom.