CodeGRITS: A Research Toolkit for Developer Behavior and Eye Tracking in IDE

Ningzhi Tang[*†], Junwen An[*†], Meng Chen[†], Aakash Bansal[†]
Yu Huang[‡], Collin McMillan[†], Toby Jia-Jun Li[†]
{ntang,jan2,mchen24,abansal1,cmc,toby.j.li}@nd.edu,yu.huang@vanderbilt.edu
[†]University of Notre Dame, Notre Dame, IN, USA
[‡]Vanderbilt University, Nashville, TN, USA

ABSTRACT

Traditional methodologies for exploring programmers' behaviors have primarily focused on capturing their actions within the Integrated Development Environment (IDE), offering limited view into their cognitive processes. Recent emergent work started using eye-tracking techniques in software engineering (SE) research. However, the lack of tools specifically designed for coordinated data collection poses technical barriers and requires significant effort from researchers who wish to combine these two complementary approaches. To address this gap, we present CodeGRITS, a plugin specifically designed for SE researchers. CodeGRITS is built on top of IntelliJ's SDK, with wide compatibility with the entire family of JetBrains IDEs to track developers' IDE interactions and eye gaze data. CodeGRITS also features various practical features for SE research (e.g., activity labeling) and a real-time API that provides interoperability for integration with other research instruments and developer tools. The demo video is available at https://youtu.be/d-YsJfW2NMI.

1 INTRODUCTION

Tracking developers' programming behavior provides valuable insights into how they engage in the software development process [9, 14, 19], and helps evaluate and improve the usability of programming language features and tools in software engineering (SE) research [11, 26]. Traditional approaches focus mainly on tracking developers' interactions with the integrated development environment (IDE), such as keystrokes, code changes, and IDE-specific commands [11, 27]. However, while these approaches can identify "what a programmer did," they are limited in explaining "why they did it." Previous research relies mainly on surveys and interviews to understand what developers were thinking and why they made certain decisions [15]. However, these approaches are susceptible to recall bias and may not capture developers' self-consciousness [7].

To bridge this gap, in recent years, researchers have started investigating the use of eye tracking to understand the cognitive processes of developers during software development such as program comprehension [23], debugging [25], and code review [3]. Eye tracking involves recording the developers' eye gaze data, i.e., the locations on the screen that the developers are looking at, while programming [22]. According to the "eye-mind hypothesis," the eye fixations (i.e., spatially stable gazes that last for 200 to 300 ms [21]) and other eye movements (e.g., saccades, blinks) are closely related to visual attention of users and the amount of cognitive processing [13]. This hypothesis has been validated in previous studies in psychology [16] and human-computer interaction [6, 24]. Furthermore, by analyzing eye gaze data, researchers can facilitate downstream SE tasks, such as automated code summarization [1, 17].

Therefore, it becomes crucial for a tool that, in addition to tracking programmers' IDE interactions, also tracks their eye gaze data to understand their cognitive processes. Some tools exist to track programmers' eye movements [5, 12] or capture their interactions with the IDE [11, 19, 27]. Notably, iTrace [12], focuses on tracking eye movement data and has been implemented as plugins in several popular IDEs, e.g., Visual Studio and Eclipse. But support for the JetBrains IDEs (e.g., InteiiJ IDEA, PyCharm), which have increased popularities in the industry and community[1][2], is lacking. Moreover, existing tools lack support to simultaneously record multiple forms of behavioral data. This inability hampers researchers' ability to conduct comprehensive studies that integrate various aspects of programmer behavior, such as eye fixations and IDE interactions, into a unified study.

In this paper, we present CodeGRITS[3], a plugin for JetBrains IDEs (e.g., IntelliJ IDEA, PyCharm, etc.) that aims to address the challenges discussed above. CodeGRITS is built on top of IntelliJ Platform Plugin SDK and uses the Tobii Pro SDK to record the eye gaze data, which could track the developers' IDE interactions and eye gaze data simultaneously. Similar to iTrace, CodeGRITS could map the eye gaze data to the specific locations (i.e., line, column) and tokens in the source code. In addition, CodeGRITS also performs an upward traversal of the abstract syntax tree (AST) for each gaze to understand its hierarchical structure. All collected data are stored locally in comprehensible formats that allow for further analysis.

Compared to previous tools like iTrace, CodeGRITS provides several extra features that cater to the specific needs of empirical SE researchers. First, a built-in screen recorder provides additional details about the developers' programming behavior and could be used to validate the eye tracking data. Second, CodeGRITS offers the functionality to add customizable labels pre-set by the researchers during tracking, to mark the developers' activities (e.g., finished debugging a bug). Finally, CodeGRITS provides a real-time data access API that allows integration with other research instruments and developer tools.

To summarize, our paper makes the following contributions:

(1) CodeGRITS, a new open-source plugin that tracks the developers' IDE interactions and eye-tracking data simultaneously during development workflows.

[*]Both authors contributed equally to this research.

[1]https://survey.stackoverflow.co/2022
[2]https://www.jetbrains.com/lp/devecosystem-2022/java/
[3]CodeGRITS stands for **G**aze **R**ecording & **I**DE **T**racking **S**ystem

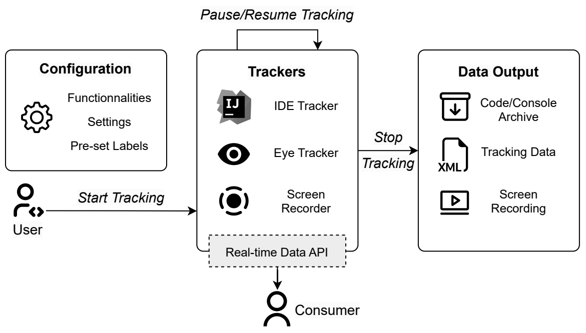
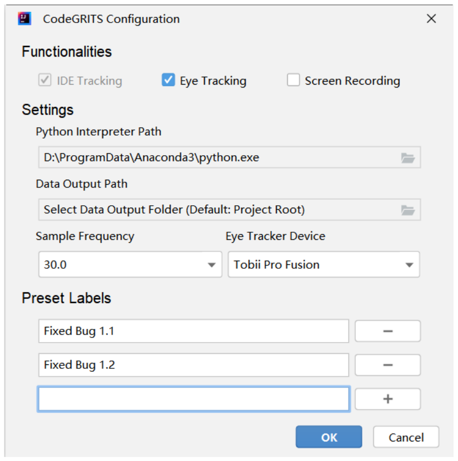


Figure 1: Overview of CodeGRITS.



Functionalities

- [✓] IDE Tracking
- [✓] Eye Tracking
- [ ] Screen Recording

Settings

Python Interpreter Path: D:\ProgramData\Anaconda3\python.exe

Data Output Path: Select Data Output Folder (Default: Project Root)

Sample Frequency: 30.0

Eye Tracker Device: Tobii Pro Fusion

Preset Labels

Fixed Bug 1.1

Fixed Bug 1.2

Figure 2: Configuration panel of CodeGRITS.

(2) CodeGRITS implements several extra features, such as a screen recorder, customizable labels, and a real-time data API, to fulfill the needs of empirical SE researchers.

(3) CodeGRITS provides wide compatibility with different JetBrains IDEs to map gazes to source code tokens for all IDE-supported programming languages.

2 OVERVIEW

Figure 1 illustrates the architecture of CodeGRITS, which consists of three components: (1) Configuration, (2) Trackers, and (3) Data Output. A typical workflow begins with the user configuring the settings in the Configuration section, followed by the activation of trackers to monitor specific interactions, e.g., IDE interactions and eye movements. Finally, the collected data is processed and presented in different output formats.

The documentation of CodeGRITS is available at https://codegrits.github.io/CodeGRITS/, which includes the usage guide, the download link, and the data format. The source code is available at https://github.com/codegrits/CodeGRITS.

2.1 Configuration

CodeGRITS offers a GUI-based Configuration panel, as shown in Figure 2. Users could set the following three types of configurations: Functionality, Settings, and Preset Labels.

2.1.1 *Functionalities.*

Users can select the trackers they want to use, including IDE Tracker, Eye Tracker, and Screen Recorder. Our plugin currently supports Tobii Pro eye-tracking devices due to its popularity in the eye-tracking community. If a compatible eye-tracking device is not available, CodeGRITS would use the mouse cursor as a substitute for eye gaze data.

2.1.2 *Settings.*

Users can configure the following settings: (1) The Python interpreter path that is used for Eye Tracker (discussed in Section 3.2); (2) The output directory for the collected data; (3) The sample frequency of Eye Tracker. The range depends on the eye-tracking device; (4) The eye-tracking device to use. The mouse is also available as a substitute.

2.1.3 *Preset Labels.*

Users are able to pre-set some labels here which could be used to mark the developers' semantic activities that cannot be captured by explicit IDE interactions. For example, when users intend to mark the time when participating in a research study on debugging, they could pre-set a label named "Bug Fixed 1.1" or "Bug Fixed 1.2" and perform "Add Label" action during tracking to mark the time when the bug is fixed. The label is also recorded in the output data via IDE Tracker.

2.2 Trackers

Considering the needs of different users, CodeGRITS comes with three trackers for various scenarios: (1) IDE Tracker, (2) Eye Tracker, and (3) Screen Recorder. After configuration, users can start the tracking process by clicking the "Start Tracking" button. The tracking process could be stopped by clicking the "Stop Tracking". Users could also pause/resume the tracking process.

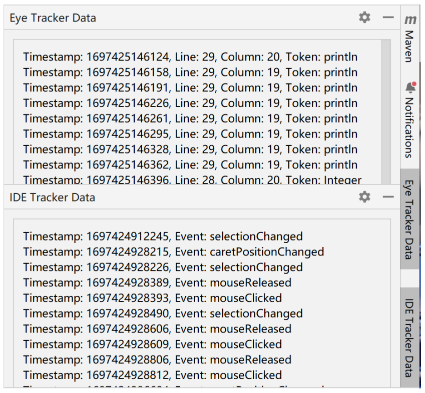
2.2.1 *IDE Tracker.*

IDE Tracker could track a wide range of IDE interactions. A sample of them is shown in Appendix A. `<actions>` part consists of IDE-specific features. These include clipboard features like `EditorPaste`, `EditorCut`; run features like `RunClass`, `Stop`, `ToggleLineBreakpoint`, `Debug`; navigation features like `Find`, `GoToDeclaration`, `ShowIntentionActions`; and much more advanced IDE features like `CompareTwoFiles`, `ReformatCode`.

In addition to IDE-specific features, IDE Tracker also tracks file system operations like `FileOpened`, `SelectionChanged`; character typing events; mouse events like `MouseMoved`, `MousePressed`; text fragment selection events; caret position events; and visible area events. Detailed information on them is shown in CodeGRITS Documentation. All tracked data are saved in an XML file with attributes (e.g., timestamp, file path) for further analysis. A *real-time archive mechanism* is also implemented to archive the whole code files when they are changed, and the console output during the development process.

2.2.2 *Eye Tracker.*

The workflow of Eye Tracker is divided into three steps: (1) connect to the eye-tracking device and receive raw data, which includes the coordinates of the eye gaze points, pupil diameters of both eyes and their validity; (2) map the coordinates of raw gazes within the text editor to specific locations in the code (i.e., file path, line and column number); (3) infer the source code tokens that each gaze point is focusing on, as well as perform a bottom-up process to traverse the AST structures of the tokens. An example of the gaze data is shown in Appendix A.



```
Timestamp: 1697425146124, Line: 29, Column: 20, Token: println
Timestamp: 1697425146158, Line: 29, Column: 19, Token: println
Timestamp: 1697425146191, Line: 29, Column: 19, Token: println
Timestamp: 1697425146226, Line: 29, Column: 19, Token: println
Timestamp: 1697425146261, Line: 29, Column: 19, Token: println
Timestamp: 1697425146295, Line: 29, Column: 19, Token: println
Timestamp: 1697425146328, Line: 29, Column: 19, Token: println
Timestamp: 1697425146362, Line: 29, Column: 19, Token: println
Timestamp: 1697425146396, Line: 28, Column: 20, Token: Integer
```

```
Timestamp: 1697424912245, Event: selectionChanged
Timestamp: 1697424928215, Event: caretPositionChanged
Timestamp: 1697424928226, Event: selectionChanged
Timestamp: 1697424928389, Event: mouseReleased
Timestamp: 1697424928393, Event: mouseClicked
Timestamp: 1697424928490, Event: selectionChanged
Timestamp: 1697424928606, Event: mouseReleased
Timestamp: 1697424928609, Event: mouseClicked
Timestamp: 1697424928806, Event: mouseReleased
Timestamp: 1697424928812, Event: mouseClicked
```

Figure 3: Real-time data output panel.

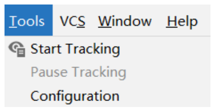


Figure 4: CodeGRITS actions in Tools menu.

2.2.3 Screen Recorder. One feature of Code-GRITS is the ability to record the screen during the tracking process. Screen Recorder captures everything on the screen and saves the capture to a video. The plugin records the timestamp of each frame, which can be used to synchronize the screen recording with other tracking data to facilitate analysis.

2.2.4 Real-time Data API. CodeGRITS provides an API for accessing IDE Tracker and Eye Tracker data in real-time, opening possibilities for researchers to integrate CodeGRITS into a multi-step data collection pipeline or develop applications using the post-processed data of CodeGRITS, such as real-time visualization. Figure 3 illustrates a working example of the usage of real-time data API, which outputs formatted Eye Tracker and IDE Tracker data in real-time into the IDE side panel.

3 IMPLEMENTATION

CodeGRITS is designed as a plugin for JetBrains IDEs due to their extensive popularity in the developer community. Our implementation is based on the official IntelliJ Platform Plugin SDK[4], as it allows us to leverage the extensibility of IntelliJ Platform with built-in APIs. The SDK is also compatible with all JetBrains IDEs such as IntelliJ IDEA, PyCharm, and CLion.

Following the guidelines of IntelliJ Platform Plugin SDK, every user-initiated action (i.e., start/stop tracking, pause/resume tracking, add label, and open configuration panel) is implemented as Java classes that extend the abstract `AnAction` class, so that they are added to IDE menus and toolbars, as shown in Figure 4. Trackers are passed into action classes as member variables, whose states are controlled via the overridden `actionPerformed` method. For instance, when the user clicks on "Start Tracking", `actionPerformed` is invoked. The method then loads the user's configuration to properly instantiate the enabled trackers and signal their activation.

[4]https://plugins.jetbrains.com/docs/intellij/welcome.html

3.1 IDE Tracker

IDE Tracker is implemented by registering listeners in the IntelliJ Platform such as `EditorMouseListener` and`SelectionListener`. Each time a specific event is triggered, the corresponding listener would be notified and the event would be recorded in the output data. The most important listener is `AnActionListener`, which tracks IDE-specific features as shown in the `<actions>` part of IDE tracking data (Appendix A). The real-time archive mechanism of IDE Tracker is implemented via `DocumentListener` interface.

3.2 Eye Tracker

We use the Tobii Pro SDK for Python[5] to collect eye tracking data and use Java `ProcessBuilder` to call the Python script to collect data. The Python interpreter path is specified in the configuration. After receiving raw data from the eye-tracking device, Eye Tracker in CodeGRITS would first compute the coordinates of each gaze relative to the top-left corner of the visible code editor. Then, the coordinates would be mapped to the specific locations in the code file (i.e., line and column) via `xyToLogicalPosition()` method of the `Editor` interface of IntelliJ Platform Plugin SDK. Next, the concrete source code tokens that the gaze points focusing on would be computed by `findElementAt()` method of the `PsiFile` interface. PSI stands for Program Structure Interface[6], which represents the underlying model of JetBrains IDEs to parse the AST of the code. Finally, Eye Tracker would iteratively use the `getParent()` method of the `PsiElement` interface to perform bottom-up traversal of AST structures of the tokens.

3.3 Screen Recorder

We use the native `Robot` class to capture screenshots between fixed time intervals, and `AWTSequenceEncoder` class in jcodec[7], a lightweight encoder library to encode screenshots into frames and generate the video.

3.4 Real-time Data API

The core of the API is Java's `Consumer` interface, which enables flexibility and extensibility. The user can implement a custom function that takes an `Element` object in package `org.w3c.dom` as input and `void` as output. Whenever a new XML element is created (e.g., IDE Tracker detected an IDE interaction), the element will be passed into the user-created function for processing. Its full documentation and usage are elaborated on CodeGRITS Documentation.

3.5 Support for Multiple IDEs and Languages

CodeGRITS supports major JetBrains IDEs, e.g., IntelliJ IDEA, PyCharm, WebStorm. Its Eye Tracker can compute the tokens and perform upward traversal of AST of all programming languages supported by each IDE. For example, in IntelliJ IDEA, the Java and Kotlin IDE, Eye Tracker could understand the AST of Java, Kotlin, Groovy, etc, while in PyCharm, the Python IDE, it could understand the AST of Python. Besides, some languages are supported by multiple IDEs, such as HTML, CSS, JavaScript, XML, etc. CodeGRITS can traverse the AST structure for the gaze on all of them.

[5]https://developer.tobiipro.com/python/python-getting-started.html
[6]https://plugins.jetbrains.com/docs/intellij/psi.html
[7]https://github.com/jcodec/jcodec

4 USE CASES

4.1 Understanding Developer Behavior and Cognition

The data collected by CodeGRITS provide a fine-grained source of information for quantitative analysis of programmers' software development process. For instance, in a previous study [25], we used an earlier version of CodeGRITS to collect data from the process of 9 programmers to debug AI-generated code. Each data collection session lasted approximately 120 minutes, and the data was used to understand their behavior and cognition patterns.

CodeGRITS could also be used by SE researchers to collect data for their studies in a wide range of tasks e.g., program comprehension, software traceability, and code review.

4.2 Context-aware Programming Support

Context-aware computing is a paradigm in which the behavior of the application is adapted to the current context of the user [8]. CodeGRITS tracks the developers' interactions with the IDE and the code, as well as the developers' eye gaze data, which form a rich source of context information about them—the developers' current behavior focus, and cognitive load. Inferring the developers' states from the tracked data lays the foundation for providing personalized programming support, potentially improving productivity and reducing their cognitive load.

Furthermore, the reduced cost and improved user-friendliness of eye-tracking devices increase the versatility of CodeGRITS, as it can easily be integrated into nonlaboratory settings, i.e., a natural development environment, and facilitate various usages.

5 LIMITATION AND FUTURE WORK

There are four main limitations of CodeGRITS. First, CodeGRITS currently only supports Tobii eye-tracking devices. However, the source code of CodeGRITS is provided for the community to expand its hardware support. We also plan to expand its support for other eye-tracking hardware SDKs in the future, too.

Secondly, CodeGRITS only captures the developer behavior and eye gaze data within the IDE, and cannot track outside activities such as browsing websites, reading documents, or using GitHub. We use Screen Recorder as a compensation to fill this gap.

Thirdly, Eye Tracker of CodeGRITS can only parse content within the editor to obtain tokens or AST structures, and cannot track other parts of the IDE, such as the menu bar or console.

Finally, CodeGRITS's tracking of the software development process focuses on objective syntactic information. It cannot interpret the subjective semantic aspects, such as finishing fixing a bug or completing writing a function. We developed the "Add Label" feature for CodeGRITS to complement this. In future work, we will explore methods to model these semantic aspects.

6 RELATED WORK

There exist several tools for collecting eye-tracking data in development environments [5, 12, 20], as well as eye-tracking data post-processing tools [2] and eye movement visualization tools [4, 18]. In particular, iTrace [12, 20] served as a fundamental infrastructure in the field, with researchers introducing several works [2, 4, 10] that extends iTrace's functionalities. Researchers also introduced several IDE plugins to capture IDE interactions [11, 27] as well as IDE interaction visualization tools [19].

Compared with the previous work described above, CodeGRITS exhibits advances by combining several behavior trackers and provides additional features to support future research.

7 CONCLUSION

In this paper, we present CodeGRITS, a plugin that uses IDE tracking, eye tracking, and screen recording methods to collect data from the software development process of developers. CodeGRITS is compatible with most JetBrains IDEs and all their supported programming languages. CodeGRITS also provides several additional features to facilitate the empirical needs of researchers.

REFERENCES

[1] A. Bansal et al. 2023. Towards Modeling Human Attention from Eye Movements for Neural Source Code Summarization. (2023).
[2] J. Behler et al. 2023. ITrace-Toolkit: A Pipeline for Analyzing Eye-Tracking Data of Software Engineering Studies. In *ICSE-Companion '23*.
[3] I. Bertram et al. 2020. Trustworthiness perceptions in code review: An eye-tracking study. In *ESEM '20*. 1–6.
[4] B. Clark et al. 2017. iTraceVis: Visualizing Eye Movement Data Within Eclipse. In *VISSOFT '17*. 22–32.
[5] A. Costi et al. 2020. CogniKit: An Extensible Tool for Human Cognitive Modeling Based on Eye Gaze Analysis. In *IUI-Companion '20*.
[6] D. K. Davis and F. Zhu. 2022. Analysis of software developers' coding behavior: A survey of visualization analysis techniques using eye trackers. *Comput Hum Behav Rep* (2022).
[7] M. C. Davis et al. 2023. What's (Not) Working in Programmer User Studies? *ACM Trans. Softw. Eng. Methodol.* (2023).
[8] A. K Dey. 2001. Understanding and using context. *Pers Ubiquitous Comput* 5 (2001), 4–7.
[9] G. Di Rosa et al. 2020. Visualizing Interaction Data Inside & Outside the IDE to Characterize Developer Productivity. In *VISSOFT '20*. 38–48.
[10] S. Fakhoury et al. 2021. gazel: Supporting Source Code Edits in Eye-Tracking Studies. In *ICSE-Companion '21*.
[11] Z. Gu et al. 2014. Capturing and Exploiting IDE Interactions. In *Onward! 2014*. 83–94.
[12] D. T. Guarnera et al. 2018. ITrace: Eye Tracking Infrastructure for Development Environments. In *ETRA '18*.
[13] M. A. Just and P. A Carpenter. 1976. Eye fixations and cognitive processes. *Cognitive psychology* 8, 4 (1976), 441–480.
[14] R. Minelli et al. 2015. I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time. In *ICPC '15*.
[15] H. Müller et al. 2014. Survey research in HCI. *Ways of Knowing in HCI* (2014), 229–266.
[16] R.-M. Rahal and S. Fiedler. 2019. Understanding cognitive and affective mechanisms in social psychology through eye-tracking. *J Exp Soc Psychol* 85 (2019).
[17] P. Rodeghero et al. 2014. Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers. In *ICSE '14*.
[18] D. Roy et al. 2020. VITALSE: Visualizing Eye Tracking and Biometric Data. In *ICSE-Companion '20*.
[19] M. Schröer and R. Koschke. 2021. Recording, Visualising and Understanding Developer Programming Behaviour. In *SANER '21*.
[20] T. R. Shaffer et al. 2015. ITrace: Enabling Eye Tracking on Software Artifacts within the IDE to Support Software Engineering Tasks. In *ESEC/FSE '15*.
[21] Z. Sharafi et al. 2015. Eye-tracking metrics in software engineering. In *APSEC '15*. 96–103.
[22] Z. Sharafi et al. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empir Softw Eng* 25 (2020), 3128–3174.
[23] Z. Sharafi et al. 2022. Eyes on Code: A Study on Developers' Code Navigation Strategies. *TSE '22* (2022).
[24] V. Skaramagkas et al. 2021. Review of eye tracking metrics involved in emotional and cognitive processes. *IEEE Rev Biomed Eng* 16 (2021), 260–277.
[25] N. Tang et al. 2023. An Empirical Study of Developer Behaviors for Validating and Repairing AI-Generated Code. PLATEAU '23.
[26] A. Yamamori et al. 2017. Can Developers' Interaction Data Improve Change Recommendation?. In *COMPSAC '17*.
[27] Y. Yoon and B. A. Myers. 2011. Capturing and Analyzing Low-Level Events from the Code Editor. In *PLATEAU '11*.

A EXAMPLE OUTPUT DATA

Listing 1: An example of IDE tracking data.

```
<ide_tracking>
    <actions>
        <action id="SaveAll" path="/src/Main.java" timestamp="1696214490354"/>
        <action id="RunClass" path="/src/Main.java" timestamp="1696214496053"/>
        <action id="ToggleLineBreakpoint" path="/src/Main.java" timestamp="1696214500296"/>
        <action id="GotoDeclaration" path="/src/Main.java" timestamp="1696214513473"/>
        <action id="Debug" path="/src/Main.java" timestamp="1696216129173"/>
        <action id="NewClass" path="/src" timestamp="1696217116236"/>
        <action id="RenameElement" path="/src/ABC.java" timestamp="1696217122074"/>
    </actions>
    <typings>
        <typing character="S" column="8" line="3" path="/src/Main.java" timestamp="1696216429855"/>
        <typing character="y" column="9" line="3" path="/src/Main.java" timestamp="1696216430111"/>
    </typings>
    <files>
        <file id="fileClosed" path="/src/Main.java" timestamp="1696216679318"/>
        <file id="selectionChanged" new_path="/src/ABC.java" old_path="/src/Main.java" timestamp="1696216679330"/>
    </files>
    <mouses>
        <mouse id="mousePressed" path="/src/DEF.java" timestamp="1696217839651" x="642" y="120"/>
        <mouse id="mouseReleased" path="/src/DEF.java" timestamp="1696217840187" x="642" y="120"/>
    </mouses>
</ide_tracking>
```

Listing 2: An example of tracked eye gaze data.

```
<eye_tracking>
    <gaze timestamp="1696224370377">
        <left_eye gaze_point_x="0.5338541666666666" gaze_point_y="0.17407407407407408" gaze_validity="1.0"
                  pupil_diameter="2.4835662841796875" pupil_validity="1.0"/>
        <right_eye gaze_point_x="0.5338541666666666" gaze_point_y="0.17407407407407408" gaze_validity="1.0"
                   pupil_diameter="2.7188568115234375" pupil_validity="1.0"/>
        <location column="25" line="2" path="/src/Main.java" x="820" y="150"/>
        <ast_structure token="println" type="IDENTIFIER">
            <level end="2:26" start="2:19" tag="PsiIdentifier:println"/>
            <level end="2:26" start="2:8" tag="PsiReferenceExpression:System.out.println"/>
            <level end="2:42" start="2:8" tag="PsiMethodCallExpression:System.out.println(&quot;Hello␣world!&quot;)"/>
            <level end="2:43" start="2:8" tag="PsiExpressionStatement"/>
            <level end="3:5" start="1:43" tag="PsiCodeBlock"/>
            <level end="3:5" start="1:4" tag="PsiMethod:main"/>
            <level end="4:1" start="0:0" tag="PsiClass:Main"/>
        </ast_structure>
    </gaze>
</eye_tracking>
```