VITALSE: Visualizing Eye Tracking and Biometric Data

Devjeet Roy
devjeet.roy@wsu.edu
Washington State University
Pullman, Washington

Sarah Fakhoury
sarah.fakhoury@wsu.edu
Washington State University
Pullman, Washington

Venera Arnaoudova
venera.arnaoudova@wsu.edu
Washington State University
Pullman, Washington

ABSTRACT

Recent research in empirical software engineering is applying techniques from neurocognitive science and breaking new grounds in the ways that researchers can model and analyze the cognitive processes of developers as they interact with software artifacts. However, given the novelty of this line of research, only one tool exists to help researchers represent and analyze this kind of multi-modal biometric data. While this tool does help with visualizing temporal eyetracking and physiological data, it does not allow for the mapping of physiological data to source code elements, instead projecting information over images of code. One drawback of this is that researchers are still unable to meaningfully combine and map physiological and eye tracking data to source code artifacts. The use of images also bars the support of long or multiple code files, which prevents researchers from analyzing data from experiments conducted in realistic settings. To address these drawbacks, we propose `VITALSE`, a tool for the interactive visualization of combined multi-modal biometric data for software engineering tasks. `VITALSE` provides interactive and customizable temporal heatmaps created with synchronized eyetracking and biometric data. The tool supports analysis on multiple files, user defined annotations for points of interest over source code elements, and high level customizable metric summaries for the provided dataset. `VITALSE`, a video demonstration, and sample data to demonstrate its capabilities can be found at http://www.vitalse.app.

ACM Reference Format:
Devjeet Roy, Sarah Fakhoury, and Venera Arnaoudova. 2020. VITALSE: Visualizing Eye Tracking and Biometric Data. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3382154

1 INTRODUCTION

Over the years, software engineering researchers are increasingly relying on techniques from neurocognitive science, through the use of psycho-physiological measures, in order to model the cognitive processes of developers. Eye tracking metrics such as pupil size, saccades, and fixation duration have been used in combination with other biometric measures, such as heart rate variability or EEG, to investigate cognitive processes during software engineering tasks. For example, Fritz et al. [5] combined EEG, eye tracking, and electrodermal activity (EDA) to investigate task difficulty during code comprehension.

The use of neuroimaging in the domain is still in its infancy. In his keynote at the International Conference on Program Comprehension (ICPC'19), Westley Weimer summarizes the less than a dozen papers so far published using high-resolution medical imaging technologies and highlights the "game changing" areas in program comprehension that open up opportunities for new research [11]. The papers published thus far, use either functional Magnetic Resonance Imaging (fMRI) or functional Near Infrared Spectroscopy (fNIRS) techniques. In the last couple of years, researchers started combining eye tracking and brain imaging techniques together [3, 4, 7]. However, in this novel integration of brain imaging techniques in empirical software engineering studies, researchers rely on tools developed for neurocognitive science practitioners in other domains. Thus, identifying a clear gap in the tool support available for interdisciplinary researchers in this expanding field.

For example, the studies using fMRI to evaluate program comprehension [9, 10] have had to rely on tools such as BrainVoyager[1] and SPM[2] which are helpful for the analysis of brain imaging data, but do not integrate other software engineering artifacts, which is vital for modeling cognitive process during tasks such as program comprehension. Recent studies we have conducted use a brain imaging technique, fNIRS, simultaneously with eyetracking data to precisely relate the cognitive load experienced by developers to specific areas in the source [3, 4]. In order to analyze and visualize the fNIRS data, we use fNIRSoft [1]. Support for the visualization of eyetracking data is widespread, with tools such as Ogama[3] and visualisation software offered by Gazepoint[4]. However, these tools support only static images or videos and are not able to visualize temporal gaze data on fine grained software artifacts such as source code. Without the precise mapping of eye gaze data to source code elements, researchers have historically relied on manual approximations for the analysis, which of course is not scaleable. iTrace [8] was developed to fill this gap, and is able to relate eyetracking data to specific source code elements. However, at this time, iTrace offers no temporal visualization support for eyetracking data.

To visualize temporal physiological and eyetracking data, Peitek et al. recently introduced CodersMuse [7]. To the best of our knowledge, this is the only available tool relevant to this line of research. The tool supports representation of fMRI, physiological data, and participant task metrics such as mouse clicks and task correctness. However, the tool does not provide fine grained mapping of eyetracking data to source code elements, instead rendering eyetracking data over images of source code. In addition to files that

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3382154

[1] Brain Innovation B.V., Netherlands, brainvoyager.com
[2] https://www.fil.ion.ucl.ac.uk/spm/
[3] http://www.ogama.net
[4] https://www.gazept.com/

are too long to be captured in one image, analysis for multiple files is not supported, which bars researchers from analyzing cognitive processes of developers in realistic settings.

We propose VITALSE, a tool that allows fine grained temporal visualization and analysis of combined eye tracking and biometric data. The main features of VITALSE are summarized as follows:

(1) Interactive and customizable heatmaps with combined eye-tracking and biometric data, with temporal brushing support, for time specific analysis.
(2) Multi-file support for heat map visualization on software artifacts, such as source code files or requirements.
(3) High level task specific biometric, eyetracking, and source code metric summaries.
(4) Support for creating and visualizing user defined annotations of points of interest in software artifacts.
(5) Visualization support for datasets containing only eyetracking data on both source code and textual artifacts, for researchers who do not use brain imaging data.

2 VITALSE

VITALSE is developed as a desktop application and it is Linux, MacOS, and Windows compatible. VITALSE and sample data to demonstrate its capabilities can be found at http://www.vitalse.app. To run, simply download the app and double click.

Figure 1 shows a screenshot of VITALSE. The tool has three main sections. The source code and heatmap are displayed in the center component. The radius and intensity of the heatmap in this example are set by duration, from the eyetracking data, and oxy, from the fNIRS data, respectively. The resulting heatmap colors indicate how much cognitive load was experienced over a source code element, whereas the size of the colored area is determined by the duration of the fixation. The left column contains participant and task metadata, such as task completion, duration, and treatment type. Below the metadata are metrics calculated from specified columns of input data. In this example, the top identifiers based on gaze fixation duration are listed. These metrics change temporally depending on the time window selected in the bottom row of the tool. Time windows are selected by dragging the mouse over the line-graph. In this case, the line graph displays the oxy and hbt averages from the input fNIRS data. The time window selected is near the last minute of the task.

Figure 2 displays an example of a usage scenario and how the data flows. In this case, researchers collect data from an fNIRS and an eye tracker while participants interact with source code files. Any type of biometric data can be used, and VITALSE supports the use of eyetracking data only as well, without any corresponding physiological data. VITALSE then creates a visualization of the provided synchronized biometric and eyetracking data by layering a custom heatmap over the source code files uploaded to the tool. Areas of code are identified using the line and column information from the eyetracking data and the heat map is then colored using the supplied physiological data. Intensity of the heatmap colors indicate the intensity of the physiological data, which is scaled and mapped to a color. For example, darker colors indicate higher cognitive load or longer fixation duration.

In the following subsections we explain the tool features, architecture, types of data sources supported by the tool, and steps for the data visualization.

2.1 Architecture

VITALSE is written entirely in JavaScript; it is completely platform independent and can run in most modern web browsers. The project utilizes React[5], a component based user interface library. This allows VITALSE to be completely modular and enables rapid development and integration of new components for evolving visualization needs. Although VITALSE is developed using web technologies, it is packaged as a desktop application using Electron [2] for easy distribution and mitigation of any browser compatibility issues. Electron is an open source development platform that allows for the development of desktop applications using web technologies.

The heat map used by VITALSE is custom developed using WebGL2 and the incremental Gaussian blur algorithm described by Nguyen [6]. A GPU based solution is required in order to implement the temporal brushing feature efficiently—the heatmap clears and redraws itself numerous times a second while the user is maneuvering the selected time window. Performing the rendering of the heatmap using the CPU incurs a significant cost on the responsiveness and usability of the visualization.

2.2 Data Sources

VITALSE supports four types of data files as input:

(1) Software artifacts, such as source code files or requirements as textual files, used by the participant during the experiment.
(2) Eye tracking data in the form of a flat JSON or CSV. Data from this file will be used to create the heatmap. The user can choose which data columns will be used to create the visualizations.
(3) Synchronized eyetracking and biometric data in the form of one flat JSON or CSV file.
(4) Physiological data in the form of flat JSON or CSV. This file will be used to draw visualisation of the physiological data, customizable on the data columns chosen by the user.

Users are prompted to identify which data columns to use for the following visualizations: radius, intensity, time, text line and column for the heatmap, and the y-axis keys for the biometric line graph.

2.2.1 Software Artifacts. VITALSE supports the use of multiple source code and textual files. Text files are displayed using the HTML *pre* tag, while source files are displayed using an HTML *code* tag inside the *pre* tag. Syntax highlighting for the latter is performed using highlight.js[6], a JavaScript syntax highlighting library which supports all major programming languages. Gaze data uploaded in the eyetracking data files is mapped to the line and columns specified in the uploaded source code or text files with corresponding file names.

[5] https://reactjs.org/
[6] https://highlightjs.org/

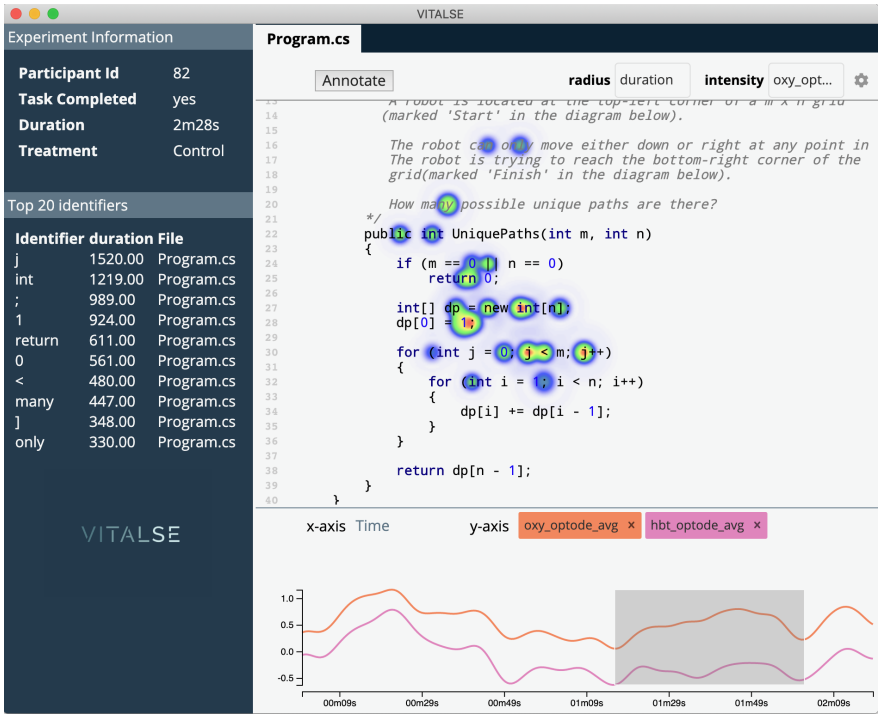

Figure 1: Screenshot of VITALSE visualizing sample fNIRS and eyetracking data.

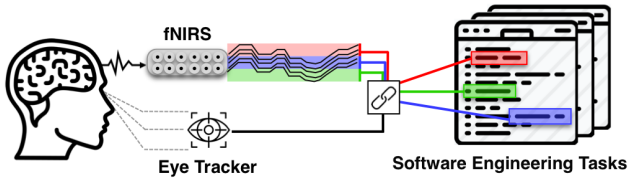

Figure 2: Data from the fNIRS and eyetracker are synchronized temporally before being visualized by VITALSE. fNIRS data is colored to indicate varying levels of cognitive load, which is then mapped to source code elements and visualized by VITALSE.

2.2.2 Eye Tracking Data. VITALSE supports data from all eye trackers, given that it includes the file name for which gaze data was collected, a timestamp, fixation duration, and the line and column numbers corresponding to the fixation. Any additional information, such as pupil size, can be added and visualized by indicating the corresponding column name in the data file. All eyetracking data pre-processing, application of fixation filters, and de-noising must be completed by the user before providing the data to VITALSE for visualization.

2.2.3 Biometric Data. VITALSE supports any type of biometric data that can be temporally mapped to eyetracking data. For example, fNIRS data, which is collected in the form of oxygenated and de-oxygenated hemoglobin from multiple optnodes, can be mapped to collected eye tracking data using timestamps. Different biometric data collection techniques might have inherent differences in the way that data should be processed or synchronized. For example, the hemodynamic response measured by fNIRS and fMRI is temporally delayed from the onset of the underlying neural activity. As a consequence, biometric data from these devices must be pre-processed before it can be synchronized to align with other forms of real time data, such as eye tracking. If using biometric data, the user must pre-process and synchronize their data with eyetracking data before providing it as input to VITALSE to be visualized. Often times, during synchronization, biometric datapoints are aggregated to fit into gaze duration windows. Therefore, in order to create visualizations for physiological data both the synchronized data, which contains biometric data mapped to eyetracking gaze points, and the original pre-processed biometric data must be provided as input.

2.3 Data Visualization

2.3.1 Eye tracking and Heatmap rendering. VITALSE provides the ability to visualize eyetracking data snapped to text content. Currently, there are no standard graphing libraries that support this capability, which we are aware of. The main challenge in providing a heatmap visualization directly over text is that font sizes, line heights, tab sizes, and other typographical properties can vary widely between computer systems, which could make the visualization inaccurate. Eyetrackers typically record eye movements and output x and y screen coordinates. To be able to visualize this data, metadata needs to be recorded while the eyetracking data is being collected, such as the screen resolution, position of the editor window, and typographic settings. Visualization becomes more complex when

eyetracking data is collected for more than one file, for example, reading through multiple modules in a software project.

To overcome this, VITALSE requires eyetracking data to be recorded in terms of line and column numbers instead of screen pixel coordinates. This can be done using a tool like iTrace [8]. The use of line and column information removes variance and allows for consistent visualization of the data. In addition, it enables the mapping of each gaze in the eyetracking data to identifiers in the source code. VITALSE uses this information to display metrics, at an identifier level of granularity, in the left column. In order to display eyetracking data that uses line and column numbers instead of screen pixel coordinates, a mapping must be created from the former to the latter. VITALSE uses Electron's Browser Selection API [7], which provides a mechanism to obtain the precise screen coordinate of user specified characters in any text rendered in the browser window. Once this mapping is generated, VITALSE draws a heatmap onto an HTML *Canvas* element overlaid on top of the source code.

2.3.2 Annotation. VITALSE allows the user to annotate points of interest in the visualised software artifacts, which can later be used in qualitative or quantitative analyses. For example, we have used VITALSE to annotate linguistic antipatterns in the source code in prior studies [4]. We have also used this feature to annotate parts of the source code that developers reported to be difficult during bug localization tasks, along with their comments. The annotation functionality provided by VITALSE allows the user to highlight parts of the code and then annotate it with multiple tags. These tags can then be exported to a JSON file, with each entry in the file containing the start and end character indices of a highlight and a list of tags the highlight was annotated with. The character indices for the highlights can be converted to line and column numbers to incorporate it with the original eyetracking data.

3 EVALUATION

We have conducted a preliminary evaluation of VITALSE while using it to visualize fNIRS and eyetracking data during two previous studies [3, 4]. As users of the tool, having the visualization was fundamental to asking appropriate follow up questions with participants. We were able to show participants areas of source code with high cognitive load, or a high duration of fixations, through different points in time of the experiment. Participants appreciated the the visual aid and were able to recall details easier while responding to our questions. The annotation feature allowed us to mark identifiers and areas of code where participants made notable remarks to include in our qualitative analysis later on. We will continue to use VITALSE for our future studies.

We plan to reach out to researchers in the domain to conduct usability surveys to asses their needs and the usefulness of the current features, incorporating their feedback into future releases. In particular, researchers that use any type of biometric and eyetracking data can use VITALSE, provided that the data can be represented in the appropriate CSV or JSON formats. As VITALSE supports eyetracking over any kind of text, not only source code, we plan to contact researchers that use any text based artifact, such as requirement documents or code review comments. Most notably, with the recent surge in support for replication and open source practices in software engineering research, VITALSE provides the opportunity for researchers to explore typically hard to obtain biometric data. Thus, we also plan to contact researchers that do not possess biometric and eyetracking equipment, but are interested in using our highly customizable tool with publicly available eye tracking and biometric data to facilitate the exploration, analysis, and visualization of experiments.

[7] https://developer.mozilla.org/en-US/docs/Web/API/Selection

4 FUTURE DIRECTIONS

Currently, VITALSE visualizes all time based physiological data in the form of a line graph. In the future we plan to customize this visualization for the specific type of physiological data. For example, displaying fNIRS and fMRI data over snapshots of specific brain regions. We also plan to support researchers performing experiments at design level by supporting different types of software artifacts, such as UML diagrams. Moreover, automatic analysis of points of interest based on user defined thresholds will be supported in future releases. Finally, we plan to investigate whether researchers would benefit from integrating the synchronization of eyetracking and physiological data into VITALSE.

5 ACKNOWLEDGMENTS

This work is supported by the NSF (award number CCF-1755995).

REFERENCES

[1] BIOPAC. 2018. fNIRSoft User Manual. https://www.biopac.com/wp-content/uploads/fnirsoft-user-manual.pdf.
[2] Electron. 2019. Electron. https://electronjs.org/.
[3] Sarah Fakhoury, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2018. The effect of poor source code lexicon and readability on developers' cognitive load. In *Proceedings of the 26th Conference on Program Comprehension*. ACM, 286–296.
[4] Sarah Fakhoury, Devjeet Roy, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. 2019. Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization. *EMSE* (2019), 1–39.
[5] Thomas Fritz, Andrew Begel, Sebastian C Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. Using Psycho-physiological Measures to Assess Task Difficulty in Software Development. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 402–413.
[6] Hubert Nguyen. 2007. *Gpu gems 3*. Addison-Wesley Professional. https://dl.acm.org/citation.cfm?id=1407436.
[7] Norman Peitek, Sven Apel, André Brechmann, Chris Parnin, and Janet Siegmund. 2019. CodersMUSE: multi-modal data exploration of program-comprehension experiments. In *Proceedings of the 27th International Conference on Program Comprehension*. IEEE Press, 126–129.
[8] Timothy R Shaffer, Jenna L Wise, Braden M Walters, Sebastian C Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: Enabling eye tracking on software artifacts within the IDE to support software engineering tasks. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 954–957.
[9] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2014. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 378–389.
[10] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. 2017. Measuring neural efficiency of program comprehension. In *ESEC/FSE*. 140–150.
[11] Westley Weimer. 2019. What goes on in your brain when you read and understand code?. In *International Conference on Program Comprehension (ICPC)—keynote*. https://web.eecs.umich.edu/~weimerw/p/weimer-icpc2019-keynote.pdf.