

# CognIDE: A Psychophysiological Data Integrator Approach for Visual Studio Code

Roger Denis Vieira  
Universidade do Vale do Rio dos Sinos  
São Leopoldo, Rio Grande do Sul  
rogervi@edu.unisinos.br  
rogerdenis@gmail.com

Kleinner Farias  
Universidade do Vale do Rio dos Sinos  
São Leopoldo, Rio Grande do Sul  
kleinnerfarias@unisinos.br  
kleinner@gmail.com

## ABSTRACT

Wearable devices capable of capturing psychophysiological data are a reality today. Recent studies indicate that the developer's cognitive indicators (e.g., level of attention and meditation) might affect code comprehension and maintenance tasks. However, current Integrated Development Environments (IDEs) and code editors like Visual Studio (VS) Code fall short of providing contextual information of cognitive indicators located throughout source code. This article proposes CognIDE, a tool-supported approach for integrating psychophysiological data related to cognitive indicators into the VS Code. CognIDE help VS code to push a step forward, providing actionable contextual information alongside the evolving source code. The CognIDE was evaluated through a survey with 6 professionals and interviews for investigating its effects on their perception of usefulness, ease of use, and intention to use in real-world settings. With a high acceptance of the professionals, the emerging results show the potential for using CognIDE to identify and prioritize the review of source code with specific cognitive indicators, mainly related to bugs or inadequate understanding of code snippets.

## CCS CONCEPTS

• **Applied computing** → **Bioinformatics**; • **Software and its engineering** → **Formal software verification**; *Integrated and visual development environments*; *Software testing and debugging*; • **Human-centered computing** → Human computer interaction (HCI).

## KEYWORDS

Data Analysis, Software Engineering, Neurosciences, Bioinformatics, Data Processing

### ACM Reference Format:

Roger Denis Vieira and Kleinner Farias. 2020. CognIDE: A Psychophysiological Data Integrator Approach for Visual Studio Code. In *34th Brazilian Symposium on Software Engineering (SBES '20)*, October 21–23, 2020, Natal, Brazil. ACM, New York, NY, USA, 6 pages. <https://doi.org/10.1145/3422392.3422453>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).  
*SBES '20, October 21–23, 2020, Natal, Brazil*

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 978-1-4503-8753-8/20/09...\$15.00  
<https://doi.org/10.1145/3422392.3422453>

## 1 INTRODUCTION

The use of wearable devices has increased in recent years. Such devices can measure and collect different types of data such as physiological data, e.g. heart rate, body temperature, and glycemic index. Personal electroencephalography (EEG) devices such as Neurosky Mindwave Mobile 2 would be an example of these wearable devices. These devices can measure the neural activity of individuals and make this data available for applications, which, in turn, provide different functionalities, such as the measurement of levels of attention and meditation [3], drowsiness, or even, enables the capability of brain-computer interaction (BCI) [2].

Recent studies [5, 6, 14] have sought to understand the brain activity of developers while they are doing software development tasks. Some studies [8, 13] aim to grasp the effects of mental states on the production of defective software artifacts. These studies show the importance of collecting psychophysiological data and relating them to tasks performed and artifacts generated by developers. Typically, software developers use integrated development environments (IDEs) to perform software development tasks. However, IDEs and wearable devices are not integrated. This makes it difficult or challenging to collect psychophysiological data while developers perform tasks in IDEs like Visual Studio (VS) Code.

Despite the need to integrate psychophysiological data into IDEs, the literature still lacks an approach capable of meeting this issue. Even worse, current IDEs and code editors like VS Code fall short of providing contextual information of cognitive indicators located throughout source code.

Murphy [11] argues that the use of contextual information as a first-class construct is an emerging research field. Once explored, it will take the tools used by software developers a step further. Such information may be applied to enhance the human capacity to deal with software development issues [11]. For example, software architects might be more effective in choosing pieces of code to be refactored when choosing those requiring greater mental effort to be understood properly. Siegmund et al. [15] also highlight the importance of understanding cognitive processes that are strictly involved in code comprehension tasks. Today, recent empirical studies [7, 10, 14] in this field have a configuration that does not match the reality of software developers. Because such studies require participants to mentally exercise small pieces of code, avoiding any type of body movement as much as possible. However, understanding source code involves not only cognitive activities, but also motor, social and human-machine interaction activities, and mainly using IDEs resources, via debugging tool and syntax highlighting, for example.

This article, therefore, proposes CognIDE, a tool-supported approach for integrating psychophysiological data related to cognitive indicators into the VS Code. CognIDE helps Microsoft VS Code to push a step forward, providing actionable contextual information alongside the evolving source code. The CognIDE was evaluated through a questionnaire with 6 professionals and interviews for investigating its effects on their perception of usefulness, ease of use, and intention to use in real-world settings. The emerging results are encouraging and show the potential for using CognIDE to support contextual information concerning cognitive indicators alongside the source code.

The remainder of the paper is organized as follows: Section 2 briefly presents a synthesis of identified similar works. Section 3 describes the proposed approach. Section 4 introduces some implementation aspects. Section 5 describes the study methodology to initially assess the proposed approach. Section 6 discusses the emerging results. Section 7 presents some concluding remarks and future work.

## 2 RELATED WORKS

Among the identified developed tools, the VITALSE [1] stands out. This tool was conceived by the need to analyze data from different devices applied in software engineering experiments, such as eye-trackers and fNIRS (functional near-infrared spectroscopy), to be loaded, synchronized, and synthesized according to the developed experiment. This feature allows the researcher to perform the data analysis in an interactive way and as a function of time, removing the need to analyze it individually and using several distinct tools.

Despite the development of some tools, there was still a demand for analysis of multimodal data from devices of different natures. In this context, CodersMUSE [12] was developed. This tool allows the integration of data collected by eye-trackers, fNIRS, fMRI (functional magnetic resonance imaging), and physiological sensors, as well as events from the individual, such as text selection and clicks. Like VITALSE, CodersMUSE also allows synchronization and visualization of data according to time, as well as its presentation in the context of the generated source code.

Unlike the observed tools, which statically presented the source code, through the images collected by the eye-tracker, and synchronized with the other psychophysiological data collected, iTraceVis [4] was developed as an extension to the Eclipse IDE. This extension, despite only synchronizing data from eye-trackers, allows the analysis of the source code used in the experiment directly in the IDE and the text format, in addition to presenting the data collected by the device in its real context.

Although the related works presented good alternatives as interactive data visualization tools, in exception of iTraceVis, which has no multimodal psychophysiological data support, these solutions presented no native integration within IDEs, neither a mechanism that contextualizes collected data with the source code non-statically.

## 3 THE COGNIDE APPROACH

Born as an extension and architecture proposal designed originally for the Microsoft Visual Studio Code editor, the CognIDE presents the capability of interacting with light-weight EEGs, enabling a

data extraction, transformation, and load to external tools, such as IDEs. Therefore, CognIDE's main objective is to provide data integration across sensor devices and IDEs, facilitating the analysis process in both academic and industrial contexts.

The data processed by CognIDE is presented visually both in the source code editor and in external dashboard tools. This resource can be used both in scientific experiments, helping in the psychophysiological processes identification presented by the developers, as a decision support tool applied in the software development process, assisting developers in the decision process in different contexts.

### 3.1 Overview of the CognIDE approach

Figure 1 introduces an overview of the intelligible, adopted the integration process of the CognIDE approach. The built-in process steps seek to collect, transform, classify and exhibit actionable contextual information related to the developer's cognitive indicators and the code snippets. That is, this process seeks to contextualize cognitive indicators by placing them next to the code snippet that was responsible for generating them. Thus, the focus is on the integration, not on the evaluation of the collected data. Each step of this process is being described as follows:

- **Step 1: Data acquisition:** This step aims at collecting the electrical signals of brain activity produced by software developers' brains while they comprehend source code or perform maintenance tasks. For this, personal wearable EEG devices, such as NeuroSky MindWave Mobile 2, were used to obtain raw values of the brain signals.
- **Step 2: Data processing:** The obtained raw data are formatted and filtered, and then their features are extracted. These features are essential to the next steps.
- **Step 3: Data storage:** At this moment, the brain signals' raw data have already been converted into cognitive indicators, using EEG devices' APIs. These indicators are sent to a storage engine, giving rise to a purported psychodataset.
- **Step 4: Psycho-data classification:** After storing the psychodataset for each developer, all datasets are tagged and their cognitive indicators are associated with the source code responsible to produce them. That is, CognIDE generates contextual information to giving context to the cognitive indicator, developer, snippets of source code, and maintenance tasks. Moreover, the CognIDE uses a context-aware approach to improve, for example, the experience of team leaders by extracting contextual metadata, allowing them to identify and prioritize the review of source code changed by developers with bug-related cognitive indicators.
- **Step 5: Psycho-KPI dashboard creation:** With the psychophysiological data sets in context, the next step is to present them to CognIDE users in a friendly way. In this sense, the CognIDE creates a psycho-KPI dashboard to represent key cognitive performance information. For this line and gauge, charts were used to display the processed cognitive indicators.
- **Step 6: Dashboard visualization:** This step involves rendering and viewing the dashboard by the CognIDE user. Typically, with tight time and projects with hundreds of modules,

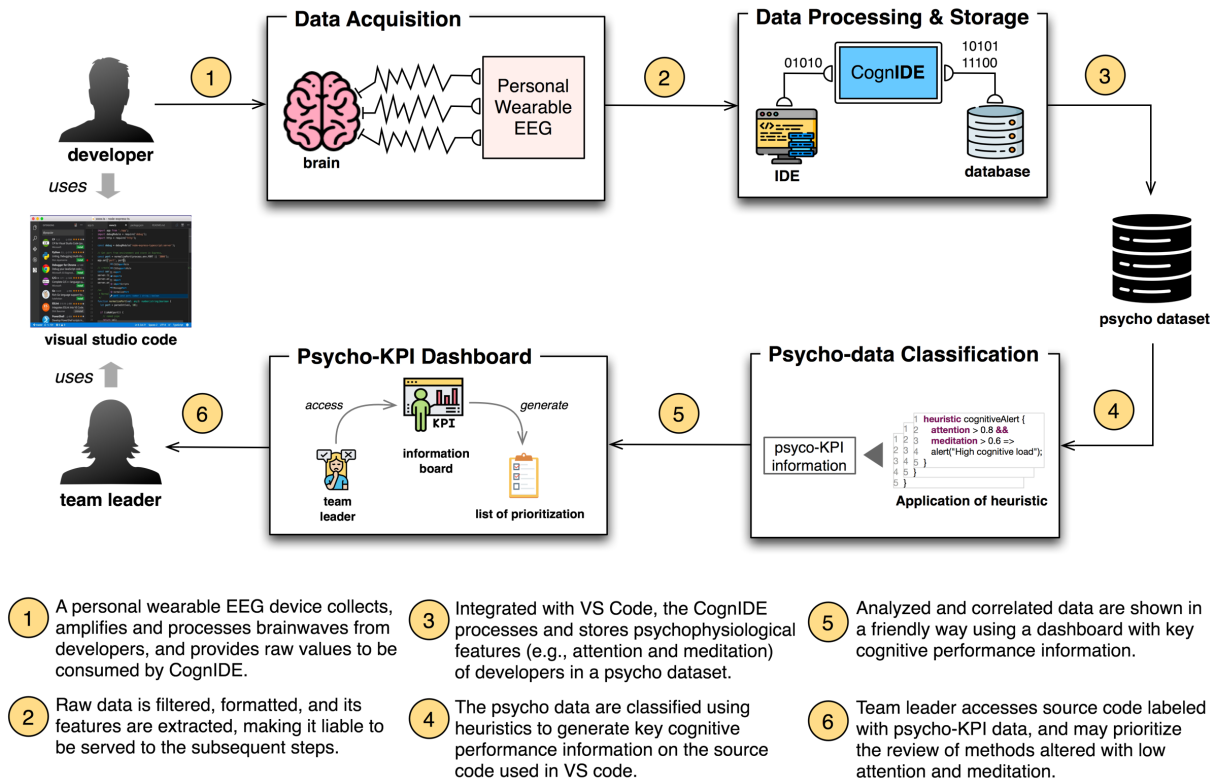


Figure 1: An overview of CognIDE operating model approach.

team leaders tend to follow their intuition when prioritizing which module to review or even refactoring. Rather than going with the gut, they can now use the psycho-KPI dashboard to make decisions based on cognitive indicators upfront. If a team leader identifies that a developer changed a snippet of source code with a low amount of attention, then he can prioritize the review of this code snapshot right away.

### 3.2 Component-based Architecture

Figure 2 introduces a component-based architecture to support the implementation of the CognIDE approach. Together, the components are responsible for implementing each step of the proposed process in Figure 1. For a better comprehension of the proposed architecture, each module is described in an abstract point-of-view, regarding its behavior rather than the applied technology as follows:

- **Data Connector:** This component receives binary data in a low-level communication layer, such as Serial Port Protocol (SPP) over Bluetooth, using the EEG-specific communication protocol, and parse the data in a high-level format such as JavaScript Object Notation (JSON), Extensible Markup Language (XML).
- **Data Adapter:** It provides an agnostic format interaction between the Data Connector and Data Processor components to reduce architectural friction in case of Data Connector change.

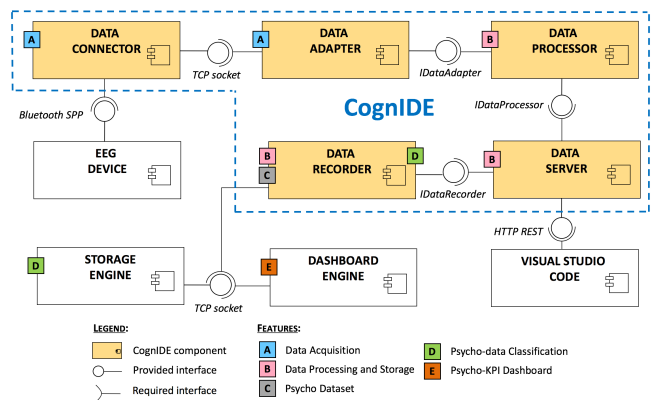


Figure 2: An overview of the component-based adopted architecture.

- **Data Processor:** It applies mathematical operations to transform and extract features from Data Connector component’s raw data into useful information that can be analyzed;
- **Data Recorder:** This component abstracts the data persistence layer, making it available for saving and retrieving already processed data.
- **Data Server:** It acts as the key component of the architecture. It receives data from the Data Connector, processes it

using the Data Processor and sends it to be displayed by the IDE. This component also receives requests from the IDE, relating the artifacts to their metrics and persisting them in the database engine.

- **IDE Plug-in:** communicating with the data server is its main responsibility, handling and presenting the exchanged data inside of the IDE, precisely alongside the source code, aiming to provide a better data visualization capable of enabling insights regarding the presented data and related code snippets.

Privacy issues remain a major concern when psychophysiological data from humans are collected. Given this issue, the proposed approach can be adapted to store and group the development team members' data according to the role they assume. In this way, the identity of the team member is preserved, ensuring privacy. After presenting an overview of the proposed process (Section 3.1) and introducing the architectural components, the following section discusses the implementation aspects of the prototype of the CogniDE approach.

## 4 IMPLEMENTATION ASPECTS

This section aims to introduce the implementation aspects, reporting the decisions taken to enable the development of a prototype<sup>1</sup>. Section 4.1 of the proposed approach presents an initial proposal of the psycho-data dashboard discussing how the main architectural components were implemented in terms of technology used. Section 4.2 presents the Visual Studio Code extension to support psychophysiological metrics located throughout the source code.

### 4.1 A Proposal of Psycho-data Dashboard

Figure 3 presents an initial proposal of psycho-data dashboard. The native psychophysiological metrics of Attention and Meditation presented were randomly generated to simulate the real data collected by the Neurosky Mindwave Mobile 2 EEG device, and not represent the real state of any specific subject. The approach were adopted for illustrative purposes only, since the main objective of this study is to assess the acceptance of the tool and the impact of presenting the data along with the source code.

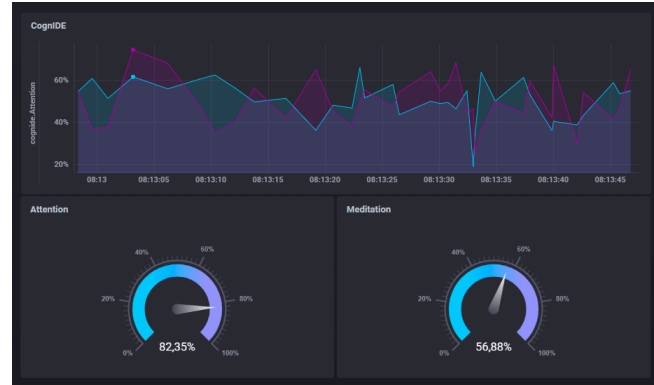
The Data Connector component (Figure 2) was accomplished using the ThinkGear Connector, a free option available on the market and that supports the Mindwave Mobile devices, a product line of light-weight EEG devices produced by NeuroSky company and wide-adopted in Brain-Computer Interaction projects and emerging neuroscience experiments.

The Data Server, namely **CogniDE Server**, was developed using the server-side JavaScript engine NodeJS. This choice was based on the fact that NodeJS's capabilities of handling requests without blocking the main thread (called *event loop*), and its event-driven support, as well. All transactions between IDE and Data Server was performed through JSON over HyperText Transport Protocol (HTTP), managed by the NodeJS's HTTP library, Express.

As a mechanism for storing processed data, InfluxDB, a time series database, was adopted. This sort of database allows a high rate of insertion operation, in addition facilitating the analysis of values oscillation over time.

<sup>1</sup><https://github.com/rogerdenisvieira/cognide-prototype>

The data stored into InfluxDB was consumed by Chronograf dashboard engine, which was responsible for presenting the processed data in an easy-to-use way (Figure 3). This choice was made upon the native integration across Chronograf and InfluxDB, once both solutions are produced by the same company.



**Figure 3:** A sample of Chronograf dashboard used to illustrate how the generated metrics are being presented for the *helloWorld.cs* artifact in a interval of 15 minutes.

### 4.2 An Extension of Visual Studio Code

CogniDE extends Visual Studio Code, namely *CogniDE extension*, to present the psychophysiological data together with the code snippets, which are responsible for producing them. In this way, the data is contextualized, providing visual feedback in the code editor. The development of the CogniDE extension took place using the Microsoft Visual Studio code editor, which provides a high-level API for the development of custom extensions (or plug-ins). Figure 4 exhibits the cognitive metrics contextualized in the code. In line 1, for example, the metrics of attention and meditation were recorded, assuming the values of 40.16% and 99.27%, respectively. This means that the developer had 40.16% attention and 99.27% meditation when editing the line 1.

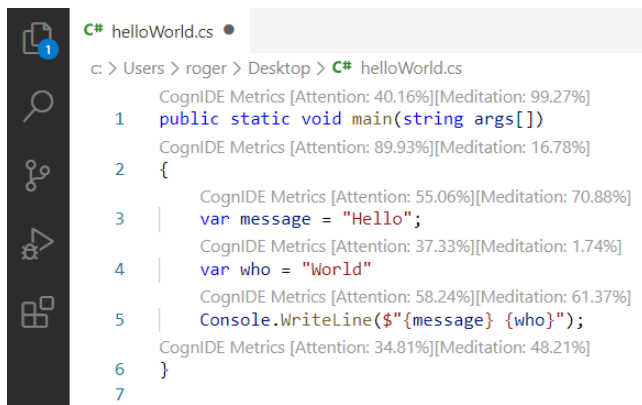
The proposed extension was implemented using the TypeScript superset and used CodeLens, a popular feature in Visual Studio Code to support actionable contextual information located throughout the source code. An important question was to define how to associate psychophysiological data with code snippets, that is, how to capture and contextualize the data. CogniDE captures the line of code changed by the developer, and then associates the psychophysiological data captured during the change of the code snippet. As the developer writes code, the CodeLens captures the typing events, sends them to the data server, which returns a set of psychophysiological metrics that, consequently, will be presented near the code snippet that has triggered the event.

## 5 METHODOLOGY

This section focuses on presenting the methodology to evaluate the CogniDE. Our assessment focused on evaluating issues considering the perception of ease of use, utility perception, attitude, and intent of behaviour by developers, regarding the CogniDE features.

**Table 1: Distribution of *Technology Acceptance Model* questionnaire responses.**

(n = 6)	I totally agree	I partially agree	Neutral	I partially disagree	I totally disagree
<i>Ease of Use Perception</i>					
I considered CognIDE easy to use	3	3	0	0	0
I considered the CognIDE easy to set it up	3	3	0	0	0
<i>Usefulness Perception</i>					
CognIDE would help to find points that the developer faced difficulties	6	0	0	0	0
CognIDE would enable the identifying of potentially defectuous code snippets	6	0	0	0	0
CognIDE would assist in code review	4	1	1	0	0
CognIDE would facilitate to decide across code refactoring prioritization	6	0	0	0	0
<i>Behavioral Intention to Use</i>					
I would use CognIDE as a code review tool	5	1	0	0	0
I would use CognIDE as a decision support tool in code refactoring prioritization	5	1	0	0	0



```

C# helloWorld.cs
c: > Users > roger > Desktop > C# helloWorld.cs
CognIDE Metrics [Attention: 40.16%][Meditation: 99.27%]
1 public static void main(string args[])
CognIDE Metrics [Attention: 89.93%][Meditation: 16.78%]
2 {
CognIDE Metrics [Attention: 55.06%][Meditation: 70.88%]
3     var message = "Hello";
CognIDE Metrics [Attention: 37.33%][Meditation: 1.74%]
4     var who = "World"
CognIDE Metrics [Attention: 58.24%][Meditation: 61.37%]
5     Console.WriteLine($"{message} {who}");
CognIDE Metrics [Attention: 34.81%][Meditation: 48.21%]
6 }
7

```

**Figure 4: Psychophysiological metrics being presented alongside the source code.**

This investigation sought to explore the following research questions: “RQ1: Is CognIDE an alternative as a support tool for the developers?”

## 5.1 Experimental Design

This phase has been characterized by: (1) the selection of the subjects, (2) the execution of the experimental process, and (3) the post-experiment data collection.

As subjects, six professionals who worked in the software development industry in various types of projects and experienced the use of different programming languages were selected by convenience and ease of access.

For the execution of the experimental process, the following steps were adopted by the participants:

- **Step 1:** Log in a remote computer running Visual Studio Code and the CognIDE server;
- **Step 2:** Access the Chronograph dashboard where the metrics are presented;
- **Step 3:** Perform an analysis based on your own perception;
- **Step 4:** In Visual Studio Code, open a snippet of source code previously implemented and interpreted the metrics presented based on your own perceptions.

Once the experimental process execution was accomplished, a set of three questionnaires were applied as the post-experimental data collection:

- **Step 1: Characterization of participants:** A sociodemographic questionnaire was applied, aiming to provide an overview of the developer’s profile;
- **Step 2: Application of TAM:** To measure the acceptance level, the participants fulfilled the contextualized version of *Technology Acceptance Model* (TAM) questionnaire [9];
- **Step 3: Interview:** The authors performed a structured interview with the selected participants, collecting feedback for aspects that have not been covered by the previous questionnaires.

It is worth noting that such metrics used in the experimental phase are not intended to assess the relationship between these and the source code produced, neither quantity any real psychophysiological state of the subjects during the experiment.

## 6 EMERGING RESULTS

Table 1 presents the collected results after applying the TAM questionnaire.

**Participant’s profile.** After applying the experimental and feedback collection phases, in a group of six participants, it was possible to verify that their half was composed of individuals aged between 20 and 29 years old, while the second half was composed of those aged between 30 and 39 years old. Mostly Software Developers (50%), followed by Software Engineers, Tech Leads, and Software Architects (16.7% each). Regarding their work experience, the majority (83.3%) had more than 6 years of acting in Software Development. When asked about their belief that the software development process can be affected by human factors, and not entirely technical ones, there was unanimity (100%) of agreement, a response that was repeated when asked if the use of psychophysiological data could be related to the quality of the code produced.

**Results from the TAM questionnaire.** Through the application of TAM, it was possible to evaluate the perceived ease of use, perceived utility, and behavioral intention to use about CognIDE. As shown in Table 1, participants agree that CognIDE is easy to use and set up (50% fully agree and 50% partially agree). There was unanimity in agreeing that the tool would help to identify points of

difficulty in the code by the developers, defective snippets of code and facilitate the decision how to refactoring the code (100% totally agree in each aspect), although there is some resistance about its usefulness in code review (66.66% totally agree, 16.67% partially agree and 16.67% are neutral). As for the intention of future use, the participants demonstrated the use as a code review tool, as well as a data provider to assist in prioritizing code refactoring (83.33% totally agree and 16.67% partially agree, in each aspect).

**Interview findings.** Regarding the interview, it was possible to verify that there was unanimity in the acceptance of the CognIDE Project by the participants, mainly in its usage as a novel tool capable of help in estimating task sizes, since it brings metrics that can be related to the complexity level of the proposed activities. Another aspect brought by the interviewees was the possibility of identifying smells in the code based on certain psychophysiological patterns, similar to what occurs in tools such as Coverity and SonarQube. Finally, even though the participants showed concerns about privacy in the collection of psychophysiological data to be used as a metric, they expressed acceptance regarding the tool, both in improving the software development process and in their productivity.

## 7 CONCLUSION AND FUTURE WORKS

This article introduced a tool-supported approach for integrating psychophysiological data into the Visual Studio Code. The CognIDE was evaluated through interviews and a questionnaire of technology acceptance with 6 professionals. Despite being a preliminary assessment, the acceptance obtained by the subjects encourages the implementation of a richer version of CognIDE in terms of new features and the development of future studies aiming to evaluate it across a bigger amount of developers.

As future works, the authors are planning to perform a controlled experiment to assess the impact of language features, such as Lambda expressions and syntax sugar, on the cognitive load of developers when they perform maintenance tasks. The proposed approach will allow the authors to capture the cognitive load while the developer changes code snippets. Furthermore, metrics for the coarser-grained block of code will be introduced in the CognIDE. In this sense, are being planned for the usage of a Language Server to improve the granularity level of the metrics. For instance, alongside a method or iteration blocks as well.

## 8 ACKNOWLEDGMENT

This work was carried out with the support of the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES), under financing code 001 and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), code 313285 / 2018-7.

## REFERENCES

- [1] Venera Arnaudova, Sarah Fakhoury, and Devjeet Roy. 2020. VITALSE : Visualizing Eye Tracking and Biometric Data. In *Demonstrations in 42 International Conference on Software Engineering*. 1–4.
- [2] Annushree Bablani, Damodar Reddy Edla, Diwakar Tripathi, and Ramalingaswamy Cheruku. 2019. Survey on brain-computer interface: An emerging computational intelligence paradigm. *Comput. Surveys* 52, 1 (2019). <https://doi.org/10.1145/3297713>
- [3] Kimberly Chu and Chui Yin Wong. 2015. Player’s attention and meditation level of input devices on mobile gaming. *Proceedings - 2014 3rd International Conference on User Science and Engineering: Experience. Engineer. Engage, i-USER 2014* October (2015), 13–17. <https://doi.org/10.1109/IUSER.2014.7002669>
- [4] Benjamin Clark and Bonita Sharif. 2017. ITraceVis: Visualizing Eye Movement Data Within Eclipse. *Proceedings - 2017 IEEE Working Conference on Software Visualization, VISSOFT 2017* 2017–Octob (2017), 22–32. <https://doi.org/10.1109/VISSOFT.2017.30>
- [5] J. Duraes, H. Madeira, J. Castelhana, C. Duarte, and M. Castelo Branco. 2016. WAP: Understanding the Brain at Software Debugging. *Proceedings - International Symposium on Software Reliability Engineering, ISSRE (2016)*, 87–92. <https://doi.org/10.1109/ISSRE.2016.53>
- [6] Benjamin Floyd, Tyler Santander, and Westley Weimer. 2017. Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017* (2017), 175–186. <https://doi.org/10.1109/ICSE.2017.24>
- [7] 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. *Proceedings - International Conference on Software Engineering 1* (2014), 402–413. <https://doi.org/10.1145/2568225.2568266>
- [8] Thomas Fritz and Sebastian C. Muller. 2016. Leveraging Biometric Data to Boost Software Developer Productivity. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (2016)*, 66–77. <https://doi.org/10.1109/saner.2016.107>
- [9] Nikola Marangunić and Andrina Granić. 2015. Technology acceptance model: a literature review from 1986 to 2013. *Universal Access in the Information Society* 14, 1 (2015), 81–95. <https://doi.org/10.1007/s10209-014-0348-1>
- [10] Sebastian C. Müller and Thomas Fritz. 2016. Using (bio)metrics to predict code quality online. *Proceedings - International Conference on Software Engineering 14-22-May- (2016)*, 452–463. <https://doi.org/10.1145/2884781.2884803>
- [11] Gail C. Murphy. 2019. Beyond integrated development environments: adding context to software development. In *IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 73–76.
- [12] Norman Peitek, Sven Apel, Andre Brechmann, Chris Parnin, and Janet Siegmund. 2019. CodersMUSE: Multi-modal data exploration of program-comprehension experiments. *IEEE International Conference on Program Comprehension 2019-May (2019)*, 126–129. <https://doi.org/10.1109/ICPC.2019.00027>
- [13] Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. 2015. Real-time monitoring of neural state in assessing and improving software developers’ productivity. *Proceedings - 8th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2015* (2015), 93–96. <https://doi.org/10.1109/CHASE.2015.28>
- [14] 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. *Proceedings - International Conference on Software Engineering 1* (2014), 378–389. <https://doi.org/10.1145/2568225.2568252>
- [15] Janet Siegmund, Norman Peitek, André Brechmann, Chris Parnin, and Sven Apel. 2020. Studying Programming in the Neuroage: Just a Crazy Idea? *Communiation of the ACM* 63, 6 (May 2020), 30–34. <https://doi.org/10.1145/3347093>