


A tool-supported approach to integrate cognitive indicators into the Visual Studio Code

Roger Vieira

(PPGCA, University of Vale do Rio dos Sinos, São Leopoldo, RS, Brasil
rogervi@edu.unisinos.br)

Kleinner Farias

(PPGCA, University of Vale do Rio dos Sinos, São Leopoldo, RS, Brazil
 <https://orcid.org/0000-0003-1891-3580>, kleinnerfarias@unisinos.br)

Abstract: Wearable devices capable of capturing psychophysiological data have emerged as a tangible reality. Recent academic investigations emphasize the pivotal role of developers' cognitive indicators—such as attention levels and cognitive load—in influencing their effectiveness in understanding and managing code-related tasks. However, existing Integrated Development Environments (IDEs) and code editors, such as Visual Studio (VS) Code, lack comprehensive contextual information on cognitive indicators alongside source code. This article, therefore, introduces CognIDE, a novel tool-supported methodology aimed at seamlessly integrating psychophysiological data linked to cognitive indicators into VS Code. Addressing this crucial gap, CognIDE enriches VS Code by offering actionable contextual cues alongside dynamic source code. The evaluation of CognIDE, involving a survey with six industry professionals and in-depth interviews, examined its perceived utility, ease of use, and real-world applicability. Encouragingly, professionals demonstrated high acceptance, indicating CognIDE's potential to identify and prioritize code segments with specific cognitive indicators, notably related to bugs or code comprehension issues. This underscores CognIDE's promise in improving code review processes.

Keywords: Biometrics, Visual Studio Code, Cognitive Indicators, Empirical Study, Integrated Development Environments

Categories: H.3.1, H.3.2, H.3.3, H.3.7, H.5.1

DOI: 10.3897/jucs.<SubmissionNumber>

1 Introduction

The use of wearable devices has become popular in recent years [Niso et al. 2023]. These devices can measure and collect different types of psychophysiological data and biometrics [Menzen et al. 2021], including heart rate, cognitive activity, body temperature, and glycemic index. Neurosky Mindwave Mobile¹ would be an example of these personal electroencephalography devices. The current wearable devices can measure the neural activity of individuals and make them available in applications, allowing access to data on the level of attention and meditation [Chu & Wong 2014], sleepiness, or even enabling the brain-computer interaction (BCI) capacity [Bablani et al. 2019].

Recent studies [Sharafi et al. 2021, Siegmund et al. 2020, Floyd et al. 2017, Duraes et al. 2016] sought to understand the brain activity of developers while performing software development tasks. Sharafi et al. (2021) [Sharafi et al. 2021] reported that there is a

¹ Neurosky: <http://neurosky.com/>

distinction in neural representation when reading source code and natural language text. It is possible to classify the type of task performed considering the brain activity. EEG devices have been explored within the realm of software engineering practices [Menzen et al. 2023, Segalotto et al. 2023, Weber et al. 2021]. Menzen et al. (2023) [Menzen et al. 2023] introduce an approach utilizing machine learning and EEG-derived biometric data to recommend software tasks tailored to a developer's skill set, enhancing task execution efficiency and reducing negative emotional states.

Segalotto et al. (2023) [Segalotto et al. 2023] use EEG data to reveal that while developers invest less cognitive effort to comprehend instructions in modular code compared to non-modular code, the temporal effort increases in modular code without significantly improving the rate of correct code comprehension, thereby emphasizing the need to refine modularization practices to align with developers' cognitive processes. Weber et al. (2021) [Weber et al. 2021] introduces a thorough overview of existing works using brain and/or autonomic nervous system activity measurements to investigate software engineering phenomena. Other studies [Fritz & Muller 2016, Radevski et al. 2021] explored the effects of mental states on the production of defective software artifacts. These studies showed 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 even challenging to collect psychophysiological data while developers perform tasks in IDEs like Microsoft Visual Studio (VS) Code.

Despite the need to integrate psychophysiological data into IDEs, the current literature still lacks an approach capable of addressing this issue. Even worse, current IDEs and code editors like VS Code fall short of providing contextual information on cognitive indicators located throughout the source code. Prior empirical investigations [Murphy 2019, D'Avila et al. 2020] have suggested that leveraging contextual information in software development projects and overall software development practices can enhance correctness during software maintenance activities while reducing the effort required to execute these tasks. Murphy (2019) [Murphy 2019] argues that the use of contextual information as a first-class construct is an emerging research field and can take the current tools used by developers one step further. Such contextual information of cognitive indicators might enhance the human capacity to deal with software development issues [Murphy 2019]. 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 properly understood. Siegmund et al. (2020) [Siegmund et al. 2020] also highlight the importance of understanding cognitive processes that are strictly involved in code comprehension tasks. Today, recent empirical studies [Fritz et al. 2014, Siegmund et al. 2014, Muller et al. 2016] in this field have a configuration that does not match the reality of software developers. Because such studies require participants to perform tasks involving small pieces of code mentally, avoiding any body movement as much as possible. However, understanding source code involves both cognitive activities and motor, social, and human-machine interaction activities, mainly using IDE resources via debugging tools and syntax highlighting, for example.

The primary novelty of our article lies in integrating biometric data, specifically psychophysiological indicators, into the software development process, particularly within Integrated Development Environments (IDEs) like Visual Studio Code. While previous research has explored the impact of cognitive indicators on developers' efficacy in comprehending and maintaining code-related tasks, our work introduces CognIDE, a novel tool-supported methodology designed to integrate such psychophysiological

data with the code editing environment seamlessly. This integration fills a critical gap in existing IDEs, which often lack comprehensive contextual information concerning cognitive indicators alongside source code. By introducing CognIDE, we propel IDEs toward a more enriched developmental framework, providing developers with actionable contextual cues alongside the dynamic source code. Through a rigorous evaluation involving industry professionals and in-depth interviews, our study demonstrates the considerable potential of CognIDE in effectively identifying and prioritizing segments of source code presenting specific cognitive indicators, thereby enhancing code review processes and improving overall software development productivity. The emerging results are encouraging and show the potential for using CognIDE to support contextual information, like cognitive indicators and IDE events, alongside the source code, thus enabling such data to be analyzed and related to the code snippets that originated them.

After scrutinizing the current literature (Section 3), we highlight two worth-investigating research problems mitigated in this work:

- *Inadequate contextual information integration in IDEs:* Before CognIDE, there was a lack of tool-supported approaches that effectively integrated psychophysiological data, such as cognitive indicators, into IDEs like Microsoft Visual Studio Code. This absence of integration hindered the provision of actionable contextual information alongside the source code, posing challenges in analyzing and correlating such data with the corresponding code snippets.
- *Limited understanding of the effects of cognitive indicators on software development:* The absence of tools like CognIDE led to a gap in understanding how cognitive indicators impact software development within IDEs. By evaluating CognIDE's effects on professionals' perceptions of its usefulness and ease of use, the study aims to address this gap by exploring how integrating cognitive indicators and IDE events alongside source code can potentially enhance developers' workflows and decision-making processes.

Our work paves the way for a more enriched developmental framework that provides actionable contextual cues, thus improving the human-computer interaction (HCI) aspect of code comprehension and maintenance tasks. Moreover, the successful evaluation of CognIDE among industry professionals highlights its potential to enhance code review processes. This means that professionals recognized that our approach can identify source code snippets presenting specific cognitive indicators. This can improve the HCI experience by mitigating bugs and enhancing developers' comprehension of code snippets within the IDE environment.

This work extends our previous work [Vieira & Farias 2020] in some aspects. First, this study includes more detailed discussions (Section 7), in which we identify some practical implications for industry and academia and outline future research directions and challenging questions worthy of investigation by the scientific community. Secondly, the proposed component-based architecture was revisited and improved by describing each component more properly. Moreover, each section of our article underwent substantial enhancements and refinements to ensure comprehensive improvement. This article presents additional discussion, identifies open challenges and trends, and outlines key underlying issues that must be tackled in future investigations.

The remainder of the paper is organized as follows. Section 2 presents the main concepts required to grasp the proposed approach. Section 3 describes the related work. Section 4 introduces the CognIDE approach. Section 5 describes how the proposed

approach was evaluated. Section 6 introduces our initial results. Section 7 introduces an additional discussion, highlighting practical implications and future research directions. Section 8 discusses some actions taken to mitigate threats to the validity of our results. Finally, Section 9 outlines some initial conclusions and future research directions.

2 Background

This section presents the key concepts to require understanding the proposed approach.

2.1 Psychophysiological Metrics and Data

The psychophysiological data² refers to the set of metrics related to the human body's physiological processes that appear as answers to external stimulus from specific situations. Metrics might be explored by correlating their variations to identify mental states or cognitive processes of a specific subject using informational labels assigned to them when exposed to certain controlled situations [Li & Jain 1988, Andreassi 2007]. A wide array of biometrics, encompassing eye, brain, skin, and cardio-respiratory metrics, are being extensively employed in contemporary research and technological applications [Menzen et al. 2021].

Eye metrics. They can be obtained using sensor devices known as *eye-trackers*. Typically, these devices use laser beams or infrared light to measure some attributes related to the subject's eyeball, e.g., eye movement trajectory, pupil dilation, focus, eye gaze, and eye blinking frequency. When applied in specific contexts, the data collected using *eye-trackers* can indicate a set of psychophysiological processes, such as cognitive load, excitation, attention, anxiety, or even stress. Knowledge regarding such psychophysiological conditions has been applied to particular software engineering domains, such as software development process [Fritz et al. 2014], debugging strategy [Lin et al. 2016], the usability of IDEs, difficulty in new languages learning, and others [Hejmady & Narayanan 2012].

Brain metrics. The neural activity of the human brain generates a potential difference along the cranial surface, as a result of the communication between the chains of neurons. The generated electrical manifestations can be captured using an electroencephalography (EEG) device [Fritz & Muller 2016, Andreassi 2007]. Researchers have identified the possibility of translating the neural activity and its voltage fluctuations in a wave spectrum grouped by different frequency bands, being able to be related to a respective set of activities (Table 1). Alpha wave (α), for instance, often can be observed in subjects who are in states of meditation or relaxation, being attenuated as physical or mental activity intensity increases [Andreassi 2007, Fritz & Muller 2016]. Electrodermal activity, as well as skin's surface temperature measurement, has already been correlated with psychological processes, such as excitation and some specific emotions [Fritz & Muller 2016].

As well as voltage fluctuations on the cranial surface, the neural activities trigger biological processes according to the execution of both physical and mental activities, such processes as the brain blood and oxygen supply can be measured employing techniques like *functional Near-Infrared Spectroscopy* (fNIRS) and *functional Magnetic Resonance*

² In academic literature, we often find the term *biometric data*; however, in this article, the authors have adopted the term *psychophysiological data*, to avoid ambiguity across the process to assign an identity to a subject based on his biological data (aka.: *biometry*)

Table 1: Brainwaves spectrum and its frequently related activities.

Wave	Symbol	Band	Relationship
Alpha	α	8 – 12Hz	relaxation, meditation, reflection
Beta	β	12 – 30Hz	alert state, focus
Gamma	γ	31 – 120Hz	high brain activity, learning
Delta	δ	0,4 – 4Hz	deep sleep
Theta	θ	4 – 7Hz	drowsiness

Imaging (fMRI). Differently from EEG, fNIRS and FMRI are capable of generating clear images of brain activity in well-defined regions, enabling their relation to cognitive and biological processes [Fishburn et al. 2014, Li & Granić 2009, Solovey et al. 2015].

Skin metrics. The set of metrics related to a variation on the skin's surface properties, such as temperature and *Electrodermal Activity* (EDA) is known by *Skin metrics*, the last one is often found in the literature under the term *Galvanic Skin Response* (GSR). The EDA sensor devices usually collect data from skin's electric resistance as a response to some external stimulus. For instance, when a subject is in a state of excitation, sweat glands increase activities, triggering an increase in sweat production. Therefore, it reduces the skin's electrical resistance, enabling a better electrical current conductivity across it [Fritz & Muller 2016, Haag et al. 2004].

Cardio-respiratory metrics. In the revision process of the studies used as theoretical grounding, four main metrics were identified and associated with the cardiorespiratory aspects presented by the subjects analyzed in the experiments: *Blood Volume Pulse* (BVP), *Heart Rate* (HR), *Heart Rate Variability* (HRV), and the *Respiratory Rate* (RR). On the one hand, the heart rate refers to the number of cardiac cycles (systolic and diastolic movements) performed by cardiac muscle for one minute; on the other hand, the heart rate variability points to time variation between two consecutive cardiac cycles. As just the HR, the respiratory rate stands for the cycles performed by the lungs to provide oxygen to the body. The blood volume pulse, also found in literature as just "pulse", measures the blood flow in human body-specific areas according to changes driven by *Sympathetic Nervous System* (SNS), especially when a subject is exposed to stressful situations [Andreassi 2007, Fritz & Muller 2016]. The presented metrics are strongly linked to cognitive load and several emotions [Fritz & Muller 2016].

2.2 Cognitive Load Theory

The human body, besides constantly providing physical resources, also called mental resources – as memory, attention, and reasoning – aims to maintain the subject's vital functions. When necessary, such resources can also be allocated in specific ways to enable the execution of harder tasks started by the subject, for instance, the resolution of logical problems they face in the software development process. The literature uses the term *Cognitive Load* [Sweller et al. 2011] regarding the number of mental resources, especially the working memory, which is used while a mental effort is necessary to perform a set of cognitive activities – both simple and complex ones.

Due to its kind, the Cognitive Load can still be split into three specialized categories:

1. **Intrinsic Cognitive Load:** It is related to the difficulty level associated with a simple instructional topic. All instructions have an inherent difficulty level associated with them, which cannot be modified by the instructional agent. However, the mental schema used to represent a task might be decomposed into smaller pieces, enabling

the possibility to be handled singly and posteriorly regrouped without compromising semantically the original schema.

2. **Extraneous Cognitive Load:** Such kind of cognitive load can be generated according to the way informational content is presented by an agent to one who will perform an activity. Different approaches to presenting information can generate distinct levels of extraneous cognitive load.
3. **Germane Cognitive Load:** Different from such other cognitive load categories, the germane cognitive load can be considered a type of *artificial* cognitive load once it results from the process of mental schemes creation used to represent a collection of instructions, such as automation, analogies, or event deconstruction and construction of concepts, intending to provide easiness in the comprehension and learning regarding a specific topic.

In short, the developer's cognitive load, for instance, can be affected by many factors, including the source code presentation in IDEs, their expertise regarding a specific programming language, their knowledge about faced errors, and their solution comprehension at all [Fritz & Muller 2016].

2.3 Integrated Development Environment

The Integrated Development Environments (IDEs) can be considered as the main set of tools used by developers for both creation tasks and source code maintenance. This set of tools holds source code editors, file navigation, compilers, debuggers, artifacts refactoring assistants, tools for syntax highlighting, such as other functionalities [Snipes & Butler 2015]. From a developer perspective, besides the expertise across the programming language that is being used and the full knowledge regarding logical aspects involved in the problem, it is essential to their domain over the tools used in the development process [Astromskis et al. 2017].

In the last years, researchers have been conducting studies about the usage of IDEs by developers, both in the software development process [Astromskis et al. 2017, Zayour & Hajjdiab 2013] and its efficacy as a debugging tool [Zayour & Hajjdiab 2013]. Usually employing eye-trackers to collect quantitative data, the authors evaluate aspects like usage easiness, user graphical interface organization and disposition, the set of available tools, such other resources according to the solution provider.

Looking for the evolution of such tools and to facilitate the development of related studies, some authors have produced articles aiming to perform a better way to integrate biometric data – such as those collected by eye-trackers – within integrated development environments in a way that data can be analyzed and related to the generated source code, or even, to the developer's interaction while executing his tasks, with no need of external tools [Guarnera et al. 2018].

2.4 Psychophysiological Data in Software Engineering

Psychophysiological data has great applicability in understanding what experiences developers are subjected to while performing their tasks during an intense, fully development-focused work cycle, as well as when facing some sort of difficulty understanding a piece of code [Fritz & Muller 2016].

In the context of software engineering, the use of psychophysiological data has been emerging as it refers to the understanding of the cognitive processes involved in software development and their correlates [Fritz & Muller 2016]. Studies in this area have explored from devising new low-cost sensor devices that are capable of reliably measuring psychophysiological signals to examining how novice and experienced developers understand source code writing [Lee et al. 2016, Szu et al. 2021] and how this can help improve software artifact production [Radevski et al. 2021].

Despite the increase in the production of studies in this area, some authors advocate the low consistency and lack of consensus presented by them, thus seeking to conduct surveys that bring a current and consolidated view around the use of psychophysiological data in the context of software engineering [Gonçales et al. 2021, Obaidellah et al. 2018]. Through the development of mapping and systematic reviews, the authors aim to identify gaps in the theme that may serve as input for the preparation of new studies able to address the problems presented.

3 Related Work

This section discusses some studies that are strictly close to the theme explored in this article. We seek to bring together studies that explore psychophysiological data in the context of software development tasks [Siegmund et al. 2020], experimental studies involving code understanding and EEG devices [Peitek 2018, Siegmund et al. 2017, Fakhoury et al. 2018], and the use of data, e.g., to predict code quality and task difficulty using machine learning techniques [Fritz & Muller 2016, Muller et al. 2016, Gonçales et al. 2021]. In addition, we paid attention to literature review studies that created an overview of the current literature [Vieira & Farias 2020, Gonçales et al. 2021, Menzen et al. 2021].

The pursuit of relevant literature was conducted using Google Scholar³. We only used Google Scholar due to its extensive coverage of scholarly literature across various disciplines, including software engineering and neuroscience. It provides access to various academic publications, including journals, conference proceedings, theses, and other scholarly sources. This makes it a comprehensive platform for retrieving relevant research articles and papers in the context of our study. The selection process primarily involved works identified through searches using the keywords “EEG” and “IDE.” This specific terminology constituted an integral aspect of the search strategy, delineating the investigation’s scope and focal point. In total, six studies were selected and carefully examined.

3.1 Analysis of the Related Work

Vieira & Farias (2020) [Vieira & Farias 2020]. This study proposed an approach (named CognIDE) designed to integrate psychophysiological data, particularly cognitive indicators, into Visual Studio (VS) Code. It addresses the limitations of existing IDEs and code editors by incorporating contextual information related to cognitive indicators dispersed within source code. Through surveys and interviews involving six professionals, the study evaluated CognIDE’s impact on perceived utility, ease of use, and real-world applicability. The proposed tool received high acceptance among professionals, indicating its potential in identifying and prioritizing code sections presenting specific cognitive

³ <https://scholar.google.com/>

indicators, notably those associated with bugs or inadequate comprehension of code snippets.

Ioannou et al. (2020) [Ioannou et al. 2020]. This study explores novel eye tracking and pupillometry approaches in software engineering and aims to visualize a developer's gaze and pupil dilation linked to source code artifacts. It proposes innovative visualizations correlating gaze patterns with code comprehension across multiple files and emphasizes the established connection between pupil dilation and cognitive load. The proposed tool seeks to provide insights into developers' cognitive processes during code comprehension tasks by associating pupil dilation with code artifacts. The article outlines a feasibility study to analyze pupil dilation in tandem with code artifacts. It outlines some initial findings, which, while promising, necessitate further elaboration and detailed discussion on practical implications and interactive visualization aspects for a comprehensive evaluation.

Arnaoudova et al. (2020) [Arnaoudova et al. 2020]. This study highlights the limitations of existing tools in integrating physiological and eye-tracking data with source code elements within empirical software engineering research. It emphasizes the scarcity of tools allowing representation and analysis of multi-modal biometric data, relying on an available tool that visualizes such data temporally but lacks direct mapping to code elements. This absence prevents a comprehensive integration of physiological and eye-tracking data with specific code artifacts, further limited by the tool's use of images and inability to support extensive or multiple code files. Proposing VITALSE, the authors aim to introduce a solution facilitating interactive visualization of synchronized biometric data for software engineering tasks. VITALSE offers customizable heatmaps, file support, user annotations, and adaptable metric summaries, enabling a deeper understanding of developers' cognitive processes interacting with software artifacts.

Peitek et al. (2019) [Peitek et al. 2019]. This study states program comprehension is a pivotal cognitive process in programming and highlights the use of multi-modal psychophysiological and neurobiological measurement methods to enhance comprehension understanding. The authors noted a lack of adequate tool support tailored for program comprehension research. They emphasize the absence of a tool enabling synchronized exploration of multiple modalities of data, specifically designed to meet the demands of research in program comprehension. CodersMUSE arises to address this identified gap, presenting a prototype designed to fulfill the essential requirement for synchronized, conjoint multi-modal data exploration in the context of program-comprehension research.

Clark & Sharif (2017) [Clark & Sharif 2017]. This study introduces iTraceVis, an eye-tracking visualization component tailored for iTrace, an Eclipse plugin capable of mapping raw eye gaze data onto corresponding source code elements based on their abstract syntax graph hierarchy. Unlike other tools, iTraceVis allows for visualizing eye-tracking data on larger source code files, surpassing the limitation of displaying only screen-fitting elements. The visualization component offers four distinct views: heat map, gaze skyline, static gaze map, and dynamic gaze map, aiming to facilitate comprehension for researchers and developers during eye tracking sessions. The authors conducted a pilot user study involving 10 senior students, presenting an existing eye-tracking developer session and assessing user interaction with iTraceVis in Eclipse. Results suggest that the visualizations effectively aid users in understanding the represented sessions, highlighting areas for potential improvement in future iterations.

3.2 Comparative Analysis and Research Opportunity

Ten Comparison Criteria (CC) have been meticulously selected to facilitate an objective comparison between the proposed work and the chosen articles. This strategic choice enables an unbiased assessment, fostering a methodical evaluation of similarities and distinctions. Such a framework is paramount in identifying research opportunities, steering clear of subjective biases that might influence the analysis. These criteria were chosen for several reasons. Primarily, they allow for a comprehensive evaluation of various facets, ensuring a holistic comparison. Secondly, they provide a standardized framework that enables an impartial and fair assessment across multiple works. Lastly, this approach has been previously validated [Menzen et al. 2021, Rubert & Farias 2021, Carbonera et al. 2023], ensuring its reliability and effectiveness in enabling objective evaluations in academic research. The description of each criterion is outlined below:

Table 2: Comparative analysis of the selected related works

Related Work	Comparison Criterion									
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
Proposed Work	●	●	●	●	●	●	●	●	●	●
Vieira & Farias (2020) [Vieira & Farias 2020]	●	●	●	●	●	●	○	○	○	○
Ioannou et al. (2020) [Ioannou et al. 2020]	●	●	●	●	●	○	○	○	○	○
Arnaudova et al. (2020) [Arnaudova et al. 2020]	●	○	●	●	●	○	○	○	○	○
Peitek et al. (2019) [Peitek et al. 2019]	●	○	●	●	●	○	○	○	○	○
Clark & Sharif (2017) [Clark & Sharif 2017]	○	●	●	●	●	●	○	○	○	○

Legend: ● Completely meets ○ Partially meets ○ Does not meet

1. **Analysis of Data Types (CC01):** Determines the tool's capacity to concurrently analyze diverse data types.
2. **Real-time Data Integration (CC02):** Assesses the tool's capability to integrate data in near real-time, enabling online analysis for experiments requiring immediate processing.
3. **Interactive Visualization (CC03):** Evaluates whether the tool offers interactive visualization for both collected data and associated code snippets over time.
4. **Contextual Data (CC04):** Focuses on evaluating tools providing contextual data within Integrated Development Environments (IDEs), encompassing both psychophysiological data and code snippets.
5. **Source Code in Text (CC05):** Assesses if the tool presents analyzed code snippets in text format, enabling symbol analysis without reliance on Optical Character Recognition (OCR) tools for text extraction.
6. **Integration with IDEs (CC06):** Determines if the tool allows seamless integration of data into existing IDEs, eliminating the necessity for a separate editor for conducting experiments and analyses.
7. **Takeaway Messages (CC07):** Emphasizes the key takeaway messages derived from the analysis.
8. **Challenge Statement (CC08):** Evaluates the clarity and appropriateness of the research challenge addressed by the tool.

9. **Practical Implications (CC09):** Explores the practical implications of the conducted study and its potential impact.
10. **Additional Discussion (CC10):** Examines the depth and relevance of the additional discussion provided, considering the acquired data and its implications.

Research opportunity. Table 2 compares the selected studies, showing that all criteria were not met. The following gaps were observed from the comparisons: (1) no work covered all the comparison criteria, except the proposed work; (2) the literature still lacks studies that explore the integration of psychophysiological data into source code; and (3) no study has explored textual aspects, considering the definition of takeaway messages, challenge statement, practical implications, and additional discussion. Therefore, this work seeks to provide comprehensive contextual information concerning cognitive indicators alongside source code by proposing an innovative approach and striving to describe findings and reflections that can inspire new research.

4 The CognIDE Approach

This section presents CognIDE, a tool-supported approach for integrating psychophysiological data of cognitive indicators into the Microsoft Visual Studio Code. The CognIDE presents the capability of interacting with lightweight EEGs, enabling data extraction, transformation, and load to external tools, such as IDEs. It was designed to extend the Microsoft Visual Studio Code editor. In this sense, CognIDE's main objective is to provide data integration across sensor devices and IDEs, facilitating the analysis process in both academic and industrial contexts. CognIDE presents data visually both in the source code editor and in external dashboard tools. This resource can be used in scientific experiments, helping in the psychophysiological process identification presented by the developers, as a decision support tool applied to support software development tasks, assisting developers in the decision process in different contexts.

4.1 Overview of the CognIDE approach

Figure 1 introduces an overview of the intelligible workflow 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 code snippets. This process seeks to contextualize cognitive indicators by placing them next to the code snippet responsible for generating them. Thus, the focus is on integration, not the evaluation of the collected data from wearable devices. Each step of this process is described as follows:

- **Step 1: Data acquisition:** This step aims to collect the electrical signals produced by the developer's neural activity during a source code comprehension task or debugging defects. For this, wearable EEG devices for personal use, such as NeuroSky Mindwave Mobile 2⁴ were used to acquire raw values from neural activity.
- **Step 2: Data processing:** The raw data from the acquisition phase are formatted and filtered, aiming at removing any possible noisy data, and their features are extracted using specific algorithms, thus serving as input for the next stage of the process.

⁴ <https://store.neurosky.com/pages/mindwave>

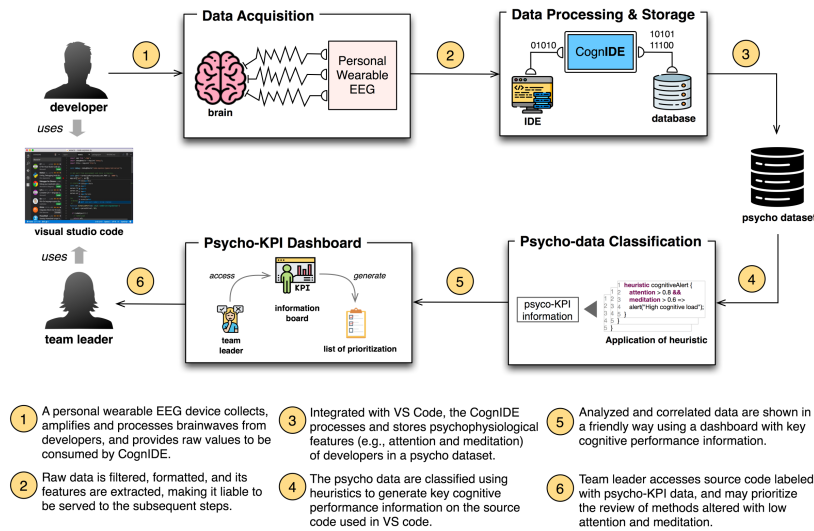


Figure 1: An overview of the CognIDE's intelligible workflow.

- **Step 3: Data storage:** At this third step, the raw data collected from neural electrical activity has already been converted into contextual indicators of neural activities., e.g., attention, meditation, drowsiness, using the set of libraries and APIs provided by the adopted EEG device. These indicators are formatted and sent to their storage in a time series database engine.
- **Step 4: Psycho-data classification:** Before the actual storage of the indicator sets collected from each observed developer, they are previously labeled with a series of metadata from the IDE being used, such as the line of code and the files that were being edited at the exact moment when the metric was collected. Therefore, CognIDE is not only responsible for collecting and processing data from the developer's neural activity, but also contextualizing metrics with the respective events related to them.
- **Step 5: Psycho-KPI dashboard creation:** With the contextual information duly included in the database, the next step is to present these in a user-friendly manner. In this regard, CognIDE allows creating a set of visualization panels (or dashboards) of the collected indicators to present relevant information to the professional responsible for the analysis.
- **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, team leaders tend to follow their intuition when prioritizing which module to review or even refactoring. Rather than going to the gut, they can 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.

After delineating the workflow aimed at gathering, transforming, classifying, and

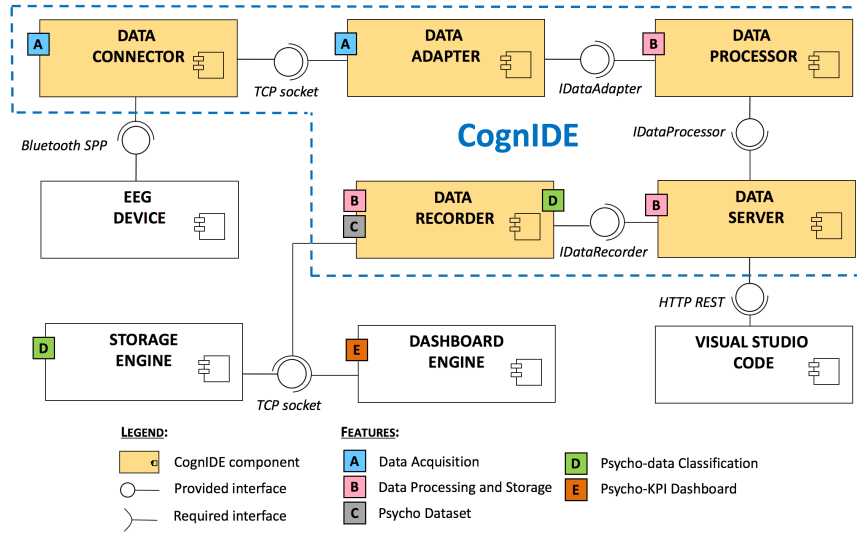


Figure 2: An overview of the component-based adopted architecture.

showcasing actionable contextual information linked to developers' cognitive indicators and code snippets, we propose a component-based architecture to support it. This architecture serves as the backbone supporting and empowering the proposed workflow.

4.2 Proposed 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 from an abstract point-of-view regarding its behavior rather than the applied technology as follows:

- **Connector:** This component receives the binary data from the device used to collect psychophysiological data, using low-level communication, such as serial communication (SSP) over Bluetooth, and applying the specific communication protocol adopted by the EEG. In addition, this component performs the serialization of data and makes it available in a high-level format that can be consumed by other applications, such as JSON and XML.
- **Adapter:** It is responsible for providing an agnostic interaction of format between the Connector and Processor components. This component aims to reduce the architectural coupling in case the Connector is replaced, allowing it to be interchangeable through configurations according to the need of the device used in the data collection.
- **Processor:** It applies a predetermined sequence of calculations on the values of the raw data provided by the Connector, to extract its features and transform them into information that can be analyzed.
- **Recorder:** Its main function is to abstract the data persistence layer, making it available for recording and reading operations of previously processed data.

- **Server:** It acts as the main component of the architecture, receiving the data from the Connector, processing it using the Processor, and sending it to be consumed and viewed by the IDE, in addition to relating it to the source code snippets produced and persisting them in the storage system through the Recorder component.
- **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 an ever-present 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 their roles. In this way, the identity of the team member is preserved, ensuring privacy. After presenting an overview of the proposed process (Section 4.1) and introducing the architectural components, the following section discusses the implementation aspects of the prototype of the CognIDE approach.

4.3 Implementation Aspects

This section aims to introduce the implementation aspects, reporting the decisions taken to enable the development of a prototype⁵. Section 4.3.1 presents the first version of the psycho-data dashboard discussing how the main architectural components were implemented in terms of used technology. Section 4.3.2 presents the Visual Studio Code extension to support psychophysiological metrics located throughout the source code.

4.3.1 A Proposal of Psycho-data Dashboard

Figure 3 displays our initial version of the psycho-data dashboard. To simulate data collection by the Neurosky Mindwave Mobile 2 EEG device, default psychophysiological metrics such as Attention and Meditation were randomly generated. This simulation was adopted solely for illustrative purposes and convenience, as the primary focus of this study is to evaluate tool acceptance and the influence of data presentation alongside source code.

The Connector component (Figure 2) was accomplished using the ThinkGear Connector, a free option available on the market 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 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 were performed through JSON over HyperText Transport Protocol (HTTP), managed by the NodeJS's HTTP library, Express.

As a mechanism for storing processed data, InfluxDB⁶, a time-series database, was adopted. This sort of database allows a high rate of insertion operation in addition to facilitating the analysis of value oscillation over time. The data stored in InfluxDB was

⁵ <https://github.com/rogerdenisvieira/cognide-prototype>

⁶ <https://www.influxdata.com/products/influxdb/>

consumed by Chronograf⁷ 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.

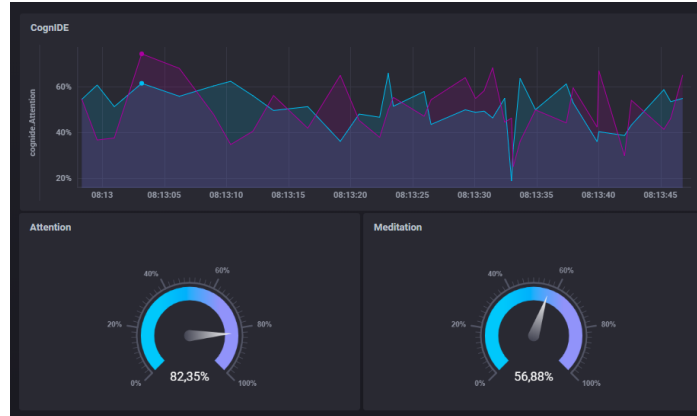


Figure 3: A sample of Chronograf dashboard used to illustrate how the generated metrics are being presented for the *helloWorld.cs* artifact in 15 minutes.

4.3.2 An Extension of Visual Studio Code

CogniDE extends Visual Studio Code, namely *CogniDE extension*, to present the psychophysiological data together with code snippets, which are responsible for producing them. This way, the data is contextualized, providing visual feedback to the code editor. The development of the CogniDE extension occurred 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 source code. In line 1, for example, the metrics of attention and meditation were recorded, assuming values of 40.16% and 99.27%, respectively. This means that the developer had 40.16% attention and 99.27% meditation when editing line 1.

The proposed extension was implemented using the TypeScript superset and used CodeLens⁸, 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 and 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.

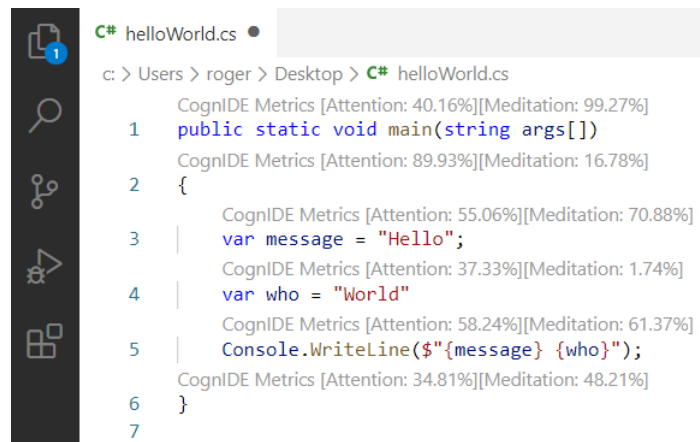
⁷ <https://www.influxdata.com/time-series-platform/chronograf/>

⁸ <https://code.visualstudio.com/api/references/vscode-api>

We bring some reflections after demonstrating the feasibility of incorporating psychophysiological data in source code into Visual Studio code in a seamless way. First, incorporating cognitive metrics contextualized within the source code, as illustrated in Figure 4, offers a novel dimension to software development practices. The specific indication of metrics like attention and meditation at particular code segments, exemplified by the attention and meditation metrics, signifies a pioneering approach in integrating neurophysiological data in software artifacts assembled throughout the software development process.

Moreover, we believe that this convergence can allow for a deeper understanding of the cognitive states of developers during code manipulation, potentially enabling a more insightful comprehension of how cognitive fluctuations impact the coding process. The obtained results not only highlight the feasibility of embedding cognitive metrics into an IDE but also puts a spotlight on the role of real-time neurophysiological data in enhancing the developers' awareness of their cognitive states during code-related tasks. These findings can hold promising implications for software developers, mainly empowering them to optimize their work patterns based on cognitive insights, thereby potentially enhancing task performance and mitigating the occurrence of errors [Menzen et al. 2023].

Secondly, these results offer a rich avenue for further exploration in the realm of neuroscience applied to software development. Researchers can explore the intricate relationship between cognitive indicators and programming activities. To further enhance the utility of such integrations, ongoing research could explore refining the precision and granularity of these metrics. The CognIDE approach can enable developers and researchers to have more detailed interpretations and practical applications within software engineering workflows.



```

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.

Table 3: 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 harmful 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

5 Evaluation

The research methodology used in our study can be categorized as a *mixed-methods research approach* [Wohlin et al. 2012], which involved the development of a tool alongside the qualitative research applying the Technology Acceptance Model (TAM) questionnaire with experts. The mixed-methods research approach combines qualitative and quantitative research methods to gather comprehensive insights into the phenomenon being studied, allowing for a more nuanced understanding by incorporating both qualitative data from the expert interviews and quantitative data from the TAM questionnaire. The study repository has the materials used throughout the research, including the questionnaires⁹.

5.1 Goal and Research Question

This study seeks to assess the effects of the CognIDE approach on the perceived usefulness, ease of use, and behavioral intention to use of our selected participants (Section 5.2). These effects are explored by considering realistic scenarios in which CognIDE can be used and adopted while developers use an IDE in real-world settings. The goal of this study is stated based on the GQM template [Wohlin et al. 2012] as follows:

**Analyze the use of CognIDE approach
for the purpose of investigating its effects
with respect to the perceived usefulness, ease of use, and behavioral intention to use
from the perspective of software developers
in the context of source code review in an IDE.**

In particular, this goal aims to explore ease of use, perceived usefulness, intention to use, influencing factors, and attitudes toward adopting the CognIDE tool. In particular, this study aims to perform a qualitative study to grasp the effects of the CognIDE approach. To do this, we sought to answer the following research question: *To what extent do software developers perceive the ease of use of the newly integrated cognitive indicators in Visual Studio (VS) Code using the CognIDE tool?*

⁹ <https://rogerdenisvieira.github.io/#resources>

5.2 Subject selection

The subject selection for this research article was based on convenience sampling [Wohlin et al. 2012]. We use convenience sampling for two reasons: considering its inherent suitability and feasibility within our qualitative assessment. First, given the nature of our study, where the proposed approach was demonstrated in real-time scenarios, convenience sampling allowed us to recruit participants readily available in academic and industry environments. That is, the participants selected for this study were deliberately chosen from the authors' professional and academic circle, ensuring the inclusion of individuals possessing highly suitable profiles for the research, thereby circumventing the selection of participants lacking academic or professional experience in using Integrated Development Environments (IDEs) and unfamiliar with the objectives of academic research. This selection approach facilitated a pragmatic assessment of the tool's functionality and usability among practitioners actively engaged in software development tasks. Secondly, employing the Technology Acceptance Model (TAM) questionnaire [Marangunić & Granić 2015] aligns with the convenience sampling strategy, enabling a comprehensive evaluation of users' acceptance and adoption of the CognIDE. We believe the pragmatic nature of convenience sampling in this context allowed for a practical and feasible means to gauge the tool's usefulness and user acceptance, thereby contributing to a more proper assessment of its real-world applicability.

In total, six participants were selected. They have worked in the software development industry in over five real projects and used different programming languages. The eligibility of the participants considered their *experience* and *availability* to participate in the stages. Our decision to involve a limited number of six participants can be explained for two reasons. First, our focus was placed on understanding the proposed approach's functionality and usability within a controlled setting and considering the limited scope of our study. We highlight involving a smaller cohort of participants facilitated a more in-depth and comprehensive examination of their interactions with the CognIDE, allowing for a detailed analysis of their feedback, perceptions, and experiences. Secondly, given the technical nature of the proposed integration with the Visual Studio code, a smaller sample size ensured meticulous attention to each participant's input and facilitated the extraction of insights and observations, ultimately contributing to the quality of the evaluation process. Additionally, a smaller sample size was considered more manageable and practical within the constraints of the study's timeline and available resources. Still, our study represents an initial exploration aimed not at definitive conclusions but at comprehensively understanding the foundational viability and coherence of the proposed approach in the context of an objective statement.

5.3 Experimental process

The experimental process involved a set of steps employed by the participants to engage with the CognIDE tool within Visual Studio Code, encompassing the following detailed procedures:

- **Step1:** Participants initiated the experimental process by logging into a remote computer housing Visual Studio Code alongside the running CognIDE server, establishing the essential environment required for the next tasks.
- **Step 2:** Upon access to the Chronograph dashboard, participants were presented with a comprehensive array of metrics derived from psychophysiological data captured

by the CognIDE tool. This dashboard facilitated the observation and analysis of cognitive indicators, offering participants an interface to comprehend and interpret the collected metrics.

- **Step 3:** Participants were tasked with performing an independent analysis based on their subjective perceptions of the presented metrics. This step allowed them to leverage their cognitive assessment and comprehension skills to interpret the data offered within the Chronograph dashboard.
- **Step 4:** Engaging with Visual Studio Code, participants were instructed to open a designated snippet of source code previously implemented. They were then prompted to interpret and discern the metrics displayed by CognIDE within the IDE, leveraging their cognitive perceptions to comprehend the information associated with the source code snippet.

Each step in this experimental process was designed to engage participants with the functionalities of the CognIDE tool in Visual Studio Code, facilitate their interaction with psychophysiological metrics, and enable subjective interpretation based on individual cognitive perceptions.

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

- **Characterization of participants:** A sociodemographic questionnaire was applied, aiming to provide an overview of the developer's profile;
- **Application of TAM:** To measure the acceptance level, the participants fulfilled the contextualized version of the *Technology Acceptance Model* (TAM) questionnaire [Marangunić & Granić 2015];
- **Interview:** The authors performed a structured interview [Wohlin et al. 2012] with the selected participants, collecting feedback on 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 them and the source code produced, nor the quantity of any actual psychophysiological state of the subjects during the experiment. The following section presents the obtained results.

6 Results

This section delves into an analysis encompassing the participant profiles, outcomes derived from the Technology Acceptance Model (TAM) questionnaire, and a discussion of the interview results.

6.1 Analysis of Participant's profile.

After applying the experimental and feedback collection phases in a group of six participants, it was possible to verify that 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. Half are Software Developers, followed by Software Engineers, Tech Leads, and Software Architects. Regarding their work experience, five of them acted in Software Development for six years or more. When asked about their belief that the software development process can be affected by human factors and not entirely technical ones, the whole sample agreed with a repeated response when asked if psychophysiological data could be related to the quality of the code produced.

6.2 Results of TAM questionnaire

Through applying TAM, it was possible to evaluate the perceived ease of use, perceived utility, and behavioral intention to use the CognIDE approach. As shown in Table 3, participants agree that CognIDE is easy to use and set up (half fully agree, and the other half 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 on how to refactor the code, although there is some resistance about its usefulness in code review (four totally agree, one partially agree and one is 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 (five totally agree and one partially agree in each aspect).

6.3 Interview findings

Regarding the interview, it was possible to verify that there was unanimity in the acceptance of the CognIDE approach by the participants, mainly in its usage as a novel tool capable of helping 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 collecting 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.

Finally, we highlight the initial nature of our findings, requiring further research to validate and expand upon these initial insights in a broader context. The primary emphasis in this evaluation was to showcase feasibility, initiate novel directions, and stimulate further directions for future investigations. In this sense, we present additional discussions on these initial findings and outline their broader implications.

6.3.1 What were your impressions of CognIDE?

We examined the interview responses to derive some qualitative findings from them. The interviews revealed multifaceted perceptions and potential implications for employing comprehensive contextual information of cognitive indicators alongside source code using the CognIDE approach. Participants exhibited varied impressions, mainly acknowledging the novelty and originality of the approach. One expressed concerns about privacy and apprehension concerning being measured (Participant 01), indicating a need for addressing potential ethical or privacy-related aspects when implementing such tools in real-world settings. Participant 02 emphasized the subjective nature of estimating cognitive indicators and the potential intellectual property implications. This subjective nature and variability in estimation might pose challenges in standardizing such measurements

across developers. Participant 03 highlighted the intriguing and enjoyable aspects of CognIDE, emphasizing its integration within the Visual Studio Code environment and the potential for generating an "interruption metric." This interruption metric could serve as a data-driven well-being indicator, shedding light on the interruptions experienced by developers during their workflow. Recognizing such metrics signals an opportunity to foster developers' well-being by considering interruptions during the coding process.

Furthermore, Participant 04 emphasized the tool's potential for enhancing productivity without harming developers. He/she indicated the widespread applicability in software engineering contexts. Finally, Participant 05 echoed sentiments regarding the innovation of the tool and its ability to generate estimates. These perceptions suggest promising implications for leveraging comprehensive contextual information of cognitive indicators alongside source code. However, they also highlight the need to address concerns related to privacy, standardization, and subjective variability in estimation while integrating such tools in software development environments. Our understanding is that while CognIDE shows promise, further refinement and consideration of these aspects are crucial to ensure its effective utilization in practice.

6.3.2 What improvements to CognIDE would you suggest?

The responses revealed constructive suggestions for refining and advancing the CognIDE approach, which has significant implications for integrating comprehensive contextual information about cognitive indicators within source code. Participant 01 advocated for a more visual representation directly in the source code. This indicates a need for a smoother integration of cognitive indicator data with IDEs. We recognize that this can help developers easily interpret and engage with cognitive indicators while working on source code.

Participant 02 proposed a multifaceted well-being approach, advocating for longitudinal data collection to enhance accuracy and provide more generalized information for developers. He or she offered a more focused perspective on well-being by suggesting that software development actions or decisions be based on cognitive indicators to improve well-being during coding sessions. This recommendation highlights the potential of cognitive indicator data not only for immediate insights but also for long-term analysis and suggestions to enhance developer well-being. Participant 03 emphasized the importance of capturing contextual noise and integrating suggestions for process improvements. This presents an opportunity for CognIDE to not only offer cognitive metrics but also to contextualize these metrics within the broader software development process, thereby contributing to process enhancement suggestions. For example, agile practices (such as sprint review ceremonies and sprint planning) could be improved by considering this cognitive aspect: how long should the ceremonies last to maintain favorable cognitive indicators for software developers in this practice?

Participant 04 suggested broadening the tool's applicability to other programming languages and highlighted the need to provide feedback while being mindful of potentially causing anxiety in developers. This recommendation underscores the importance of adaptability across various programming environments and the necessity for sensitive feedback mechanisms that consider developers' emotional well-being. Participant 05 proposed code refactorings similar to "smells" based on difficulties encountered, drawing parallels to design patterns in software development. This innovative suggestion hints at the potential for using cognitive indicators to propose specific "Neuro Refactoring Patterns." In particular, this observation opens a pathway for new research by enabling developers to identify source code anomalies [Souza et al. 2018] and subsequently

optimize such code based on mental patterns formed during the coding process. Lastly, these recommendations highlight an opportunity for CognIDE to expand its scope by offering insights into code-related patterns and proposing personalized solutions for developers' well-being and coding efficiency.

7 Additional Discussion

This section examines practical implications and future research directions based on the proposed approach and our initial results.

7.1 Practical implications for Industry and Academia

Developer efficiency and cognitive insights. CognIDE's integration of psychophysiological data into Visual Studio Code can enhance a nuanced understanding of developers' cognitive indicators, such as attention levels and cognitive load, while coding. For example, if a developer's attention level significantly drops during the coding process, CognIDE may flag this as a potential sign of code comprehension issues. This insight provides developers with real-time awareness of their cognitive state, aiding in the identification of critical moments when their focus wanes and enabling proactive interventions or breaks to reduce errors or inefficiencies. For instance, Segalotto et al. [Segalotto et al. 2023] reported that using EEG data shows promise in generating valuable cognitive insights within software development contexts. They presented compelling evidence, demonstrating that high cognitive load acts as a predictive indicator of potential software design problems and a high likelihood of misunderstanding the code.

Software quality via bug identification or task recommendation. CognIDE's ability to identify specific cognitive indicators associated with bugs or poor code comprehension offers a proactive approach to improving software quality. If a developer consistently shows increased cognitive load when working on certain code segments, CognIDE could highlight these areas during the code review process. This focuses on potential bug-prone sections, allowing developers to address errors or ambiguities before they escalate, thereby enhancing code quality and software reliability. Additionally, we can utilize biometric data, especially EEG-based biometrics, to provide more accurate recommendations for aligning software development tasks with individual developers' cognitive skills and competencies. In fact, some studies are already researching this direction. For instance, Menzen et al. [Menzen et al. 2023] outline the implementation of 4Experts, an innovative methodology that uses machine learning and biometric data obtained through EEG to recommend software development tasks suited to developers' skill sets and competencies. This approach aims to optimize task assignment by evaluating developers' cognitive load through EEG data, with the goal of reducing frustration and improving task performance.

Optimization of code review processes for improved decision-making. By leveraging CognIDE's ability to prioritize segments of code associated with specific cognitive indicators, such as instances of insufficient comprehension, code reviews become more targeted and efficient. Case developers spend a disproportionately longer time comprehending certain code snippets, CognIDE might flag these segments for deeper review. This focused approach allows teams to allocate resources more effectively, concentrating efforts on areas requiring greater attention, thereby improving code robustness and enabling better-informed decisions throughout the software development life-cycle.

7.2 Future directions and challenging aspects

Real-time cognitive monitoring for adaptive IDEs. Exploring the feasibility of integrating real-time cognitive monitoring within development environments like CognIDE presents a compelling avenue for future research. Specifically, we believe that investigating mechanisms to continuously monitor developers' cognitive indicators during coding or modeling sessions and dynamically adjust the environment based on these indicators poses a significant challenge. Designing algorithms or AI-driven systems that recognize cognitive patterns in real-time and provide adaptive interventions, such as recommending breaks or altering interface elements to align with developers' cognitive states, remains a promising yet challenging area. Menzen et al. [Menzen et al. 2023] demonstrated 83% accuracy in predicting developer expertise using machine learning and EEG data. Furthermore, leveraging cognitive indicators to guide developers toward taking breaks and relaxing, rather than continuing with erroneous code writing or misunderstanding source code, can lead to developing more effective and context-aware tools in the software development landscape.

Ethical considerations. As integrating psychophysiological data into software development tools like CognIDE progresses, it raises ethical concerns regarding data privacy, consent, and potential biases. Future research should delve into establishing ethical guidelines and frameworks governing the collection, storage, and utilization of developers' psychophysiological data in such environments. Addressing these ethical implications and ensuring developers' rights and privacy while using tools that collect sensitive cognitive data constitutes a worth-investigating challenge.

Validation across diverse development environments. Conducting extensive validation studies and generalizing the findings across diverse development environments and developer demographics represents an important future direction. Exploring how CognIDE's effectiveness translates across various programming languages, software development methodologies, and diverse developer skill sets poses a challenging yet crucial area of investigation. After conducting our study, we believe that performing robust studies across different contexts to evaluate CognIDE's efficacy and adaptability remains an imperative challenge for future research endeavors in this domain.

7.3 Empirical study, replications and human factors

A quality model for the integration of biometrics in software development. Various quality models have emerged over the past decades to assess design modeling in software engineering [?]. However, these models typically focus on software modeling in a broader context rather than specifically addressing the integration of biometric data. To advance this field, future research could extend existing quality models to encompass the unique challenges and considerations of integrating biometric data into software development processes. This extension could draw from practical insights gleaned from developers' experiences with integrating biometric data in real-world projects, as well as insights from empirical studies conducted by researchers. By leveraging evidence-based approaches, such as controlled experiments, industrial case studies, interviews, and observational studies, researchers can develop a comprehensive quality model tailored to the nuances of biometric integration. Such a model would guide developers and researchers in planning and executing empirical studies to address integration challenges effectively. Additionally, it could establish a common terminology for integration-related activities and artifacts, facilitating a systematic approach to assessing quality through qualitative and quantitative metrics. Through this structured framework, researchers can

identify and evaluate factors impacting the effort, accuracy, granularity, and scalability of biometric integration. For example, the quality model could aid developers in selecting appropriate metrics and methodologies to assess how factors such as abstraction levels, domain-specific considerations, and development or refactoring techniques influence the precision and accuracy of biometric integration approaches. Moreover, the quality model could serve as a reference framework for structuring and comparing empirical studies conducted by researchers, enhancing the reproducibility and generalizability of their findings in the burgeoning field of biometric integration in software development.

Enhancing empirical understanding on the benefits of using biometrics in software engineering. Despite the existing empirical studies, there remains a need for further research to deepen our evidence-based understanding of the advantages of incorporating biometrics into software engineering practices. For instance, conducting controlled experiments could shed light on the benefits or usefulness of particular biometrics, such as cognitive load, on revealing the efficiency of developers in producing accurate software models or source code. We can collect different biometrics of developers while performing development or maintenance tasks, but their comparative effectiveness remains largely unexplored. Moreover, there is a lack of experimental studies examining the influence of prior experience with a particular technology on developers' productivity. For instance, researchers could design controlled experiments involving participants with varying levels of experience to assess how prior familiarity with a particular IDE, for example, affects the development outcomes. These outcomes might be measured using biometrics. Such empirical investigations can uncover valuable insights into the role of biometrics in measuring or predicting the effectiveness of developers in software engineering contexts.

Understanding the impacts of biometrics on software development tasks. An important research endeavor for the scientific community focuses on understanding the effects of biometrics on various software development tasks, including source code comprehension, refactoring, and software modeling. Specifically, there is a need to examine how human cognitive processes, decision-making methods, and collaborative efforts interact with the integration of biometric data into software development workflows. It is clear that integrating biometrics involves complex activities, such as identifying patterns, resolving discrepancies, and managing interdependencies, all requiring human involvement. Therefore, exploring human factors can provide valuable insights into optimizing the biometric integration process. For instance, research could investigate the influence of different cognitive biases or heuristics on decision-making processes while integrating biometric data. Understanding how cognitive biases, such as confirmation bias or anchoring bias, affect the identification and resolution of merge conflicts can yield valuable insights into potential challenges and suggest strategies to mitigate their effects.

Seamless integration of biometrics in software engineering workflows. A pivotal research challenge in the domain of using biometrics in software development involves the creation of robust and reliable methodologies for seamlessly integrating biometric data into existing software engineering workflows. The potential benefits of incorporating biometric information, such as physiological and cognitive indicators, into software development processes are clear. However, there remains a lack of standardized approaches for effectively leveraging this data. Researchers must explore methodologies that facilitate the integration of biometric data in a way that enhances developers' understanding, productivity, and overall software quality. This requires addressing various technical, methodological, and ethical considerations, such as ensuring data accuracy and reliability, designing user-friendly interfaces for data visualization and interpretation, and

establishing privacy-preserving mechanisms to protect sensitive biometric information. Additionally, exploring ways to mitigate potential biases or disparities introduced by biometric data usage is an essential avenue for future research in this field. By tackling these challenges, the research community can pave the way for the effective use of biometrics to enhance software development processes and outcomes.

Federated learning and biometrics. A pressing research challenge at the intersection of federated learning [Antunes et al. 2022] and biometrics in the context of software development revolves around ensuring the privacy, security, and ethical implications of using sensitive biometric data in distributed learning environments. Federated learning offers opportunities for collaborative model training across decentralized devices without centralizing raw data, but integrating biometric data into this framework presents unique challenges. For example, individual data on developers' cognitive performance cannot be revealed. Ensuring the confidentiality and integrity of biometric data during the federated learning process is crucial to prevent unauthorized access, data breaches, and potential misuse. Additionally, researchers need to explore techniques for federated aggregation of biometric data that preserve individual privacy while still enabling effective model updates. Addressing concerns relating to data anonymization, differential privacy, and the robustness of federated learning models in the presence of diverse and dynamic biometric inputs is essential for building trust and acceptance of federated learning approaches in software development contexts. Moreover, investigating the ethical implications of leveraging biometric data in federated learning, including issues regarding consent, fairness, and algorithmic bias, is critical to ensure the responsible and equitable deployment of such technologies. By tackling these challenges, the research community can advance the development of federated learning techniques that harness the power of biometric data while upholding privacy and ethical standards in software development practices.

8 Threats to Validity

This section briefly discusses the strategies adopted to manage potential threats to the study's validity. In particular, we manage validity threats regarding external threats.

It refers to the validity of the results obtained in broader contexts. We seek to understand to what extent the results obtained can be extended or generalized, for example, to other IDEs (e.g., IntelliJ, NetBeans, PyCharm, and Spring Tool Suite 4), wearable devices (e.g., Emotiv EPOC), and participant profiles. In this sense, we analyzed whether the results could be maintained by making some variations. As this study has not yet been replicated, we chose to use the theory of proximal similarity to understand the degree of generalization of our results. Three criteria were defined to establish the comparisons: (1) the IDE needs to be extensible; (2) wearable devices must have an API for capturing psychophysiological data; and (3) participants with experience with different IDEs and programming languages. Given that these criteria can be observed in real project contexts, then the generalization of our results becomes feasible for similar contexts.

Additionally, decisions were made to minimize threats associated with the proposed approach. First, we selected tools and wearable devices that are widely accepted within the industry, thereby facilitating the transfer and application of the approach in other contexts. This choice ensures the approach is not tightly linked to a specific, non-extensible device or tool. Second, the component-based architecture enhances the implementation and comprehension of modules, emphasizing modularization and the connections between the main features of the approach. This facilitates the evolution of the proposed architecture by allowing for the flexible addition, modification, and removal of architectural

components. Third, the cognitive indicators utilized are generated by the wearable device, thereby minimizing potential issues when computing such indicators. Although this is an initial study that requires replication and expansion, we have established strategies to mitigate potential threats.

9 Conclusions and Future Work

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. We can outline two takeaways messages from our study. First, CognIDE represents an important step toward bridging the gap between developers' cognitive indicators and code comprehension within IDEs. This tool successfully integrates psychophysiological data into Visual Studio Code, offering contextual cues that can significantly aid in understanding and maintaining code-related tasks. Secondly, CognIDE shows promise in improving code review practices by effectively identifying code segments associated with specific cognitive indicators. Its potential in pinpointing areas related to bugs or insufficient code comprehension (based on cognitive metrics) highlights its potential to enhance code review processes, potentially leading to improved software quality. In future work, we list a series of research avenues that can be explored by the research community.

Data Volume. During the development phase of the CognIDE tool, the EEG NeuroSky MindWave Mobile 2 was utilized. This sensor features a measurement channel and a sampling rate of only 512 Hz, which is lower than other EEGs in the same market segment. Despite these specifications, the MindWave Mobile 2 can generate a large volume of data during use, especially when operating synchronously, meaning it sends the data stream to a computer with the data connector installed. This characteristic is a limiting factor both in the manipulation of such a large volume of data for proper analysis and in the required storage capacity.

Processing Capacity. Notwithstanding the resources required for the storage of the collected data, the computer on which the CognIDE tool is operating demands a high processing and memory capacity, given that the manipulation of data from the EEG takes place in real-time. This behavior is shown as a limitation given the fact that CognIDE would be operating concurrently with the other tools present on the host computer, such as the IDE itself and the development environment, competing for resources and reducing both its performance and performance of other applications.

Electroencephalography. EEG devices, especially the basic low-cost models, can suffer in different ways from environmental interference, from myoelectric noise of synergistic muscle interaction to electromagnetic interference from electronic equipment, especially those with inductive and capacitive loads. These interferences can lead to artifact formation during the data analysis, leading to erroneous results in the data collected.

Ethical and privacy aspects. Although CognIDE's evaluation did not use psychophysiological data collected from real developers, the popularization of sensor devices, in addition to the possibility brought by CognIDE in the use of such devices as input for the creation of metrics within the IDEs, from an ethical point of view and of privacy, can lead to the limitation of the applicability of the tool to only the academic

environment. Therefore, it is emphasized that such use of data within companies may violate current legislation regarding the privacy of its employees. Despite the limitations presented, the emerging results encourage exploring such limitations as opportunities for research and development of future works, aiming to evolve both the tool and the proposed approach. Therefore, the authors will explore the following main research opportunities.

Controlled Experiment. The authors plan 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, they plan to use a Language Server to improve the granularity level of the metrics. For instance, alongside a method or iteration block as well.

Device Support. In its first version, CognIDE only supports integrating psychophysiological data from EEG NeuroSky Mindwave Mobile 2. However, the architecture adopted for implementing CognIDE allows the extension of support to new devices. In future works, it is possible to work on the creation of components capable of collecting data from EMGs, Eye-trackers, EDA, among others, and integrating them in Visual Studio Code using the approach suggested by CognIDE, thus enabling the conduction of new types of controlled experiments.

Support for IDEs. As with device support, the initial version of CognIDE was designed to work with Visual Studio Code. Despite of this, its architectural flexibility allows that in future studies, new extensions are implemented to support other IDEs existing in the market.

Language Server Implementation. As mentioned in Section 4.3, both CognIDE Extension and CognIDE Server use line code change events as a trigger to trigger the collection of measurements from the EEG. Such event granularity can be worked by developing a language server capable of identifying blocks of code, such as methods, classes, or variables, to allow a more accurate generation of events. Furthermore, through the language server, it would be possible to support the syntax of any existing programming language for generating events.

Ethics Issue

The research was conducted in settings where ethics approval for survey studies was deemed unnecessary due to the nature of the investigation, which involved voluntary participation without any potential risks or sensitive information being collected from the participants.

Acknowledgement

This work was supported in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), under Grant No. 314248/2021-8, and Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS), under Grant No. 21/2551-0002051-2.

References

- [Andreassi 2007] Andreassi, J. L.: “Psychophysiology: Human behavior and physiological response,”; 5th ed. Lawrence Erlbaum Associates Publishers, 2007, ISBN: 0-805849505 (Hardcover).
- [Antunes et al. 2022] Antunes, R., Costa, C., Küderle, A., Yari, I. A., & Eskofier, B.: “Federated learning for healthcare: Systematic review and architecture proposal”; ACM Transactions on Intelligent Systems and Technology (TIST), 13(4), 2022, pp. 1-23.
- [Arnaoudova et al. 2020] Arnaoudova, V. and Fakhoury, S. and Roy, D.: “VITALSE : Visualizing Eye Tracking and Biometric Data”; 42th International Conference on Software Engineering, 2020, pp. 1-4.
- [Astromskis et al. 2017] Astromskis, S., Bavota, G., Janes, A., Russo, B., & Di Penta, M.: “Patterns of developers behaviour: A 1000-hour industrial study”; Journal of Systems and Software, 132, 2017, pp. 85-97.
- [Bablani et al. 2019] Bablani, A., Edla, D. R., Tripathi, D., & Cheruku, R.: “Survey on brain-computer interface: An emerging computational intelligence paradigm”; ACM Computing Surveys, 52(1), 2019, 1-33.
- [Carbonera et al. 2023] Carbonera et al.: “Software Merge: A Two-Decade Systematic Mapping Study”; XXXVII Brazilian Symposium on Software Engineering, 2023, pp. 99-108.
- [Chu & Wong 2014] Chu, K., & Wong, C.: “Player’s attention and meditation level of input devices on mobile gaming”; 3rd International Conference on User Science and Engineering: Experience. Engineer. Engage, i-USER 2014, October, 2014, pp. 13-17.
- [Chu & Wong 2014] Chu, K., & Wong, C.: “Player’s attention and meditation level of input devices on mobile gaming.”; In: rd International Conference on user science and engineering (i-USER), 2015, pp. 13-17.
- [Clark & Sharif 2017] Clark, B., & Sharif, B.: “ITraceVis: Visualizing Eye Movement Data Within Eclipse”; IEEE Working Conference on Software Visualization, 2017, pp. 22-32.
- [Dalcin et al. 2023] Dalcin, G., Bolzan, W., Lazzari, L., & Farias, K.: “Recommendation of UML Model Conflicts: Unveiling the Biometric Lens for Conflict Resolution”; In: XXXVII Brazilian Symposium on Software Engineering, 2023, pp. 83-88.
- [D’Avila et al. 2020] D’Avila, L. F., Farias, K., & Barbosa, J.: “Effects of contextual information on maintenance effort: a controlled experiment”; Journal of Systems and Software, 159, 2020, 110443.
- [Duraes et al. 2016] Duraes, J., Madeira, H., Castelhana, J., Duarte, C., & Branco, M.: “WAP: Understanding the Brain at Software Debugging”; International Symposium on Software Reliability Engineering, ISSRE, 2016, 87-92.
- [Fakhoury et al. 2018] Fakhoury, S., Ma, Y., Arnaoudova, V., & Adesope, O.: “The effect of poor source code lexicon and readability on developers’ cognitive load.”; In: International Conference on Software Engineering, 2018, pp. 286-296.
- [Fishburn et al. 2014] Fishburn, F. A., Norr, M. E., Medvedev, A. V., & Vaidya, C.: “Sensitivity of fNIRS to cognitive state and load”; Frontiers in Human Neuroscience, 8(1 FEB), 2014, pp. 1-11.
- [Floyd et al. 2017] Floyd, B., Santander, T., & Weimer, W.: “Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise”; IEEE/ACM 39th International Conference on Software Engineering, 2017, pp. 175-186.
- [Fritz & Muller 2016] Fritz, T., & Muller, S.: “Leveraging Biometric Data to Boost Software Developer Productivity”; 23rd International Conference on Software Analysis, Evolution, and Reengineering, 2016, pp. 66-77.
- [Fritz et al. 2014] Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., & Zuger, M.: “Using psychophysiological measures to assess task difficulty in software development.” International Conference on Software Engineering, 2014, pp. 402-413.

- [Gonçales et al. 2021] Gonçales, L. J., Farias, K., Kupssinskü, L., & Segalotto, M.: “The effects of applying filters on EEG signals for classifying developers’ code comprehension”; *Journal of Applied Research and Technology*, 19(6), 584-602.
- [Gonçales et al. 2021] Gonçales, L. J., Farias, K., & da Silva, B.: “Measuring the cognitive load of software developers: An extended Systematic Mapping Study”, *Information and Software Technology*, 136, 2021, 106563.
- [Guarnera et al. 2018] Guarnera, D. T., Bryant, C. A., Mishra, A., Maletic, J. I., & Sharif, B.: “iTrace: Eye tracking infrastructure for development environments.”; In: *Eye Tracking Research and Applications Symposium (ETRA)*, 2018, pp. 2015-2017.
- [Haag et al. 2004] Haag, A., Goronzy, S., Schaich, P., & Williams, J.: “Emotion recognition using bio-sensors: First steps towards an automatic system.”; In: *Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science*, 2004, Vol. 3068, pp. 36-48. Springer Berlin Heidelberg.
- [Hejmady & Narayanan 2012] Hejmady, P., & Narayanan, N.: “Visual attention switching patterns of programmers debugging with an IDE.”; In: *Symposium on Eye Tracking Research and Applications*, 2012, pp. 197-200.
- [Ioannou et al. 2020] Ioannou, C., Bakgaard, P., Kindler, E., & Weber, B.: “Towards a tool for visualizing pupil dilation linked with source code artifacts”; In *2020 Working Conference on Software Visualization (VISSOFT)*, 2020, pp. 105-109.
- [Lee et al. 2016] Lee, S., Matteson, A., Hooshyar, D., Kim, S., Jung, J., Nam, G., & Lim, H.: “Comparing Programming Language Comprehension between Novice and Expert Programmers Using EEG Analysis”; *16th International Conference on Bioinformatics and Bioengineering, BIBE 2016*, pp. 350-355.
- [Li & Jain 1988] Li, S. Z., & Jain, A. K. (Eds.). *Encyclopedia of Biometrics*, Second Edition. Springer US, 2015.
- [Li & Granić 2009] Li, M., & Lu, B. L.: “Emotion classification based on gamma-band EEG.”; In: *31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society: Engineering the Future of Biomedicine, EMBC*, 2009, pp. 1323-1326.
- [Lin et al. 2016] Lin, Y. T., Wu, C. C., Hou, T. Y., Lin, Y. C., Yang, F. Y., & Chang, C.: “Tracking Students’ Cognitive Processes during Program Debugging-An Eye-Movement Approach.”; *IEEE Transactions on Education*, 59(3), 2016, pp. 175-186.
- [Obaidellah et al. 2018] Obaidellah, U., Al Haek, M., & Cheng, P.: “A survey on the usage of eye-Tracking in computer programming”; *ACM Computing Surveys*, 51(1), 2018, pp. 1-58.
- [Marangunić & Granić 2015] Marangunić, N., & Granić, A.: “Technology acceptance model: a literature review from 1986 to 2013.”; *Universal Access in the Information Society*, 14(1), 2015, 81-95.
- [Menzen et al. 2021] Cabane, H., & Farias, K.: “On the impact of event-driven architecture on performance: An exploratory study”; *Future Generation Computer Systems*, 153, 2024, pp. 52-69.
- [Menzen et al. 2023] Menzen, J. P., & Oliveira, K.: “4Experts: A Task Recommendation Approach Using Machine Learning and Biometric Data.” In: *XIX Brazilian Symposium on Information Systems*, 2023, pp. 1-8.
- [Muller et al. 2016] Müller, S. C., & Fritz, T.: “Using (bio)metrics to predict code quality online.”; In: *International Conference on Software Engineering*, 2016 pp. 452-463.
- [Murphy 2019] Murphy, G.: “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)*, 2019, pp. 73-76.
- [Menzen et al. 2021] Menzen, J. P., Farias, K., & Bischoff, V.: “Using biometric data in software engineering: A systematic mapping study”; *Behaviour & Information Technology*, 40(9), 880-902. Taylor & Francis.

- [Niso et al. 2023] Niso, G., Romero, E., Moreau, J. T., Araujo, A., & Krol, L.: “Wireless EEG: A survey of systems and studies”; *NeuroImage*, 269, 2023, 119774.
- [Peitek 2018] Peitek, N.: “A neuro-cognitive perspective of program comprehension”; *International Conference on Software Engineering*, 2018, 496-499.
- [Peitek et al. 2019] Peitek, N., Apel, S., Brechmann, A., Parnin, C., & Siegmund, J.: “CodersMUSE: Multi-modal data exploration of program-comprehension experiments.” In: *IEEE International Conference on Program Comprehension*, 2019, pp. 126-129.
- [Radevski et al. 2021] Radevski, S., Hata, H., & Matsumoto, K.: “Real-time monitoring of neural state in assessing and improving software developers’ productivity”; *8th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2015, pp. 93-96.
- [Rubert & Farias 2021] Rubert, M., & Farias, K.: “On the effects of continuous delivery on code quality: A case study in industry.” *Computer Standards & Interfaces*, 81, 2022, 103588.
- [Segalotto et al. 2023] Segalotto, M., Bolzan, W., & Farias, K.: “Effects of Modularization on Developers’ Cognitive Effort in Code Comprehension Tasks: A Controlled Experiment”; *XXXVII Brazilian Symposium on Software Engineering*, 2023, 206-215.
- [Sharafi et al. 2021] Sharafi, Z., Huang, Y., Leach, K., & Weimer, W.: “Toward an Objective Measure of Developers’ Cognitive Activities.”; *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(3), 2021, 1-40.
- [Siegmund et al. 2014] Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Brechmann, A.: “Understanding understanding source code with functional magnetic resonance imaging”; *International Conference on Software Engineering*, 2014, pp. 378-389.
- [Siegmund et al. 2017] Siegmund, J., Peitek, N., Parnin, C., Apel, S., Hofmeister, J., Kästner, C., Begel, A., Bethmann, A., & Brechmann, A.: “Measuring neural efficiency of program comprehension”; *11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 140-150.
- [Siegmund et al. 2020] Siegmund, J., Peitek, N., Brechmann, A., Parnin, C., & Apel, S.: “Studying programming in the neuroage: just a crazy idea?”; *Communications of the ACM*, 63(6), 2020, pp. 30-34.
- [Snipes & Butler 2015] Snipes, W., & Butler, J.: “A Practical Guide to Analyzing IDE Usage Data.” In *The Art and Science of Analyzing Software Data*, 2015, pp. 85-138.
- [Souza et al. 2018] Sousa, L., Oliveira, A., Oizumi, W., Barbosa, S., Garcia, A., Lee, J., Kalinowski, M., de Mello, R., Fonseca, B., Oliveira, R.: “Identifying design problems in the source code: A grounded theory.”; In: *40th International Conference on Software Engineering*, 2018, pp. 921-931.
- [Solovey et al. 2015] Solovey, E. T., Afergan, D., Peck, E. M., Hincks, S. W., & Jacob, R. J. K.: “Designing implicit interfaces for physiological computing: Guidelines and lessons learned using fNIRS.”; *ACM Transactions on Computer-Human Interaction*, 21(6), 2015, 35:1-35:27.
- [Sweller 1988] Sweller, J.: “Cognitive load during problem solving: Effects on learning.”; *Cognitive Science*, 12(2), 1988, pp. 257-285.
- [Sweller et al. 2011] Sweller, J., Ayres, P., & Kalyuga, S.: “Cognitive Load Theory”; 2011, 1st ed., Springer.
- [Szu et al. 2021] Szu, H., Hsu, C., Moon, G., Yamakawa, T., Tran, B. Q., Jung, T. P., & Landa, J.: “Smartphone Household Wireless Electroencephalogram Hat”; *Applied Computational Intelligence and Soft Computing*, 2013, pp. 1-8.
- [Vieira & Farias 2020] Vieira, R. D., & Farias, K.: “CognIDE: A psychophysiological data integrator approach for Visual Studio Code”; *XXXIV Brazilian Symposium on Software Engineering*, 2010, pp. 393-398.
- [Vieira & Farias 2020] Vieira, R. D., & Farias, K.: “Usage of psychophysiological data as an improvement in the context of software engineering: A systematic mapping study”; *XVI Brazilian*

Symposium on Information Systems, 2020, pp. 1-8.

[Weber et al. 2021] Weber, B., Fischer, T., & Riedl, R.: “Brain and autonomic nervous system activity measurement in software engineering: A systematic literature review.”; *Journal of Systems and Software*, 178, 2021, 110946.

[Wohlin et al. 2012] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: “Experimentation in software engineering”; Springer Science & Business Media, 2012.

[Zayour & Hajjdiab 2013] Zayour, I., & Hajjdiab, H.: “How much integrated development environments (IDEs) improve productivity?”; *Journal of Software*, 8(10), 2013, 2425-2431.