Information and Software Technology 136 (2021) 106563

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

# Measuring the cognitive load of software developers: An extended Systematic Mapping Study

Lucian José Gonçales [a,*], Kleinner Farias [a], Bruno C. da Silva [b]

[a] Applied Computing Graduate Program (PPGCA), University of Vale do Rio dos Sinos (Unisinos), São Leopoldo, Rio Grande do Sul, Brazil
[b] Computer Science and Software Engineering Department, California Polytechnic State University (Cal Poly), San Luis Obispo, CA, United States of America

ARTICLE INFO

*Keywords:*
Cognitive load
Machine learning
Software engineering
Systematic Mapping Study

ABSTRACT

**Context:** Cognitive load in software engineering refers to the mental effort users spend while reading software artifacts. The cognitive load can vary according to tasks and across developers. Researchers have measured developers' cognitive load for different purposes, such as understanding its impact on productivity and software quality. Thus, researchers and practitioners can use cognitive load measures for solving many aspects of software engineering problems.
**Problem:** However, a lack of a classification of dimensions on cognitive load measures in software engineering makes it difficult for researchers and practitioners to obtain research trends to advance scientific knowledge or apply it in software projects.
**Objective:** This article aims to classify different aspects of cognitive load measures in software engineering and identify challenges for further research.
**Method:** We conducted a Systematic Mapping Study (SMS), which started with 4,175 articles gathered from 11 search engines and then narrowed down to 63 primary studies.
**Results:** Our main findings are: (1) 43% (27/63) of the primary studies focused on applying a combination of sensors; (2) 81% (51/63) of the selected works were validation studies; (3) 83% (52/63) of the primary studies analyzed cognitive load while developers performed programming tasks. Moreover, we created a classification scheme based on the answers to our research questions.
**Conclusion:** despite the production of a significant amount of studies on cognitive load in software engineering, there are still many challenges to be solved in this particular field for effectively measuring the cognitive load in software engineering. Therefore, this work provided directions for future studies on cognitive load measurement in software engineering.

## 1. Introduction

The cognitive load mesasures are an important developer-centric indicator in software engineering. Software developers perform tasks that demand their cognitive load constantly. In neurophysiology and educational psychology, the cognitive load term indicates the amount of load a material imposes on users' mental capacity [1,2]. In software engineering, the cognitive load term does not have a definition. However, it generally refers to users' mental effort when reading software artifacts or cognitive processing tasks. For instance, comprehending code requires that developers spend mental effort while reading source code [3,4] elements. Moreover, developers also apply cognitive load to solve math equations [5] during the interpretation of software artifacts that differ in terms of abstraction level [6]. Furthermore, the cognitive load was also applied to provide insights related to human-factors in software engineering. Thus, the usage of cognitive load data with machine learning techniques successfully predicted the experience on computer programming [7], and for identifying the difficulty level during software engineering tasks [8,9]. Another benefit is that the cognitive load can potentially have more predictive power on software engineering tasks [10] since empirical studies demonstrated that traditional software-based metrics are not able to capture developers' characteristics, thus affecting software quality and effort estimation [11]. Despite evidence on the benefits of measuring software developers' cognitive load, it is still challenging. Previous studies diverged on how to measure cognitive load and adopted different approaches [3,12]. Consequently, practitioners and researchers select and adapt cognitive load measures according to their purpose.

However, selecting measures to assess the cognitive load is not a trivial task. First, there is a lack of studies in the literature that have

* Corresponding author.
*E-mail address:* lucianj@edu.unisinos.br (L.J. Gonçales).

https://doi.org/10.1016/j.infsof.2021.106563
Received 4 July 2020; Received in revised form 25 January 2021; Accepted 23 February 2021
Available online 17 March 2021

systematically summarized and categorized the related technologies and cognitive load measures. Another problem is that, notwithstanding that some surveys and literature reviews related to the measurement of cognitive load were produced [13,14]. In particular, Gui et al. [13] reviewed brain-related measures for recognition and authentication systems. Bablani et al. [14] surveyed about technologies and measures used with Brain–Computer Interfaces (BCI). These researches do not cover the software engineering research field. The second reason would be that selecting measures to assess cognitive load is difficult due to a lack of classification through different dimensions [9,12,15]. Furthermore, the understanding of essential aspects still lacks, including sensors and measures, application of machine learning algorithms, and adoption of research methods.

Therefore, this article has two main objectives: (1) to classify primary studies about the measurement of developers' cognitive load; and (2) to pinpoint further challenges for future research. For this reason, we carried out a Systematic Mapping Study (SMS) based on guidelines that are well-recognized in literature [16–18]. We defined a systematic protocol to search in eleven electronic databases that are widely recognized for indexing peer-reviewed studies on software engineering [17,19]. Thus, we carefully carried out a filtering process on an initial search result of 4175 studies. Then, we selected 63 articles as primary studies that answered the research questions we defined for this study. We analyze and discuss the results in the remainder of this paper.

This article is an expanded and revised version of our previous work [20]. The differences between this article and the previous version are the following. First, while the study methodology is the same in this work, we expanded the search string, explicitly adding more cognitive load synonyms, such as "mental effort", and more synonyms related to "software engineering" (i.e., "program comprehension", and "software analysis" and "software deployment"). Second, we revisited the selection process, adding to it snowballing throughout the selected papers. This resulted in the selection of 63 primary studies, precisely 30 studies more than the previous version. The analysis and results implied in new findings and research gaps identified and reported. The main contributions of this article are:

- An updated classification of studies in cognitive load in software engineering based on a higher amount of primary studies and more categories concerning the previous version [20];
- A representative list of primary studies about cognitive load in software engineering;
- A new research question that classifies how the studies defined cognitive load in software engineering;
- Added discussions on the gaps and the classification results through research questions;
- Discussion on the contributions and further challenges we identified from the selected primary studies.

The remainder of this article is organized as follows: Section 2 describes underlying background concepts and related work. Section 3 presents the systematic mapping methodology we adopted. Section 4 discusses the results of the research questions. Section 5 presents an additional discussion on the findings and describes challenges for future research. Section 6 describes the measures we took to avoid bias to the validity of this research results. Finally, Section 7 presents concluding remarks and future work.

## 2. Background and related works

This section describes the basic concepts underlying this work. Section 2.1 presents the concepts of cognitive load and their types. Section 2.2 presents the concepts of Systematic Mapping Studies. Section 2.3 presents the related works.

### 2.1. Cognitive load

The studies in software engineering use the term cognitive load for referring to the mental effort users spend when reading software artifacts or while processing cognitive tasks [9,10,21]. In educational psychology, cognitive load is a term that means the amount of load that materials or activities imposes on users working memory [1]. The working memory is a mental space where users store and process elements contained in the material.

The fact is that measuring cognitive load in software engineering is important because developers can demand different mental effort. Individual factors such as age, personality traits, and expertise level differ among developers. The literature on software engineering attributes that high cognitive load generally is linked to situations where developers are mentally processing a non-trivial task [3,4,21]. On the contrary, a low cognitive load indicates that developers are solving a trivial task.

Researchers in software engineering investigated the cognitive load to analyze the impact of human factors in software engineering tasks. Studies demonstrated that biometric measurements reflect cognitive load levels [22]. The main advance about traditional metrics, such as, time of task completion and feedback formularies, is that the cognitive load measures obtained from body parts open the possibility to measure cognitive load direct from the source while developers are working on software artifacts. Studies point out that one of the benefits is predicting a bug before developers insert it into the software [23]. Another advantage is to monitor and attribute cognitive load data in relation to each timestamp of the interaction between developers and software artifacts [23,24]. The data related to each timestamp enables a more granular measure of cognitive load [25].

Many studies emerged measuring cognitive load in software engineering for various purposes. Crk and Kluthe [3] and Crk et al. [4] evidenced the difference in developers' level of expertise based on their cognitive load. For this, the authors measured the oscillations of electric pulses from the scalp using an electroencephalogram. Siegmund et al. [26] correlated the cognitive load levels with the program comprehension of developers. Siegmund et al. [26] measured the blood flow from the developer's brain through a magnetic resonance sensor while they were comprehending code.

Furthermore, many studies used machine learning techniques to classify, for example, code quality concerns [23], task's difficulty level [9], and level of expertise [7]. As this comprehends in an emerging research area, an organization in terms of sensors, metrics, machine learning techniques, purposes, tasks, artifacts, and participants are needed to organize the area. In this case, a Systematic Mapping Study methodology was the solution recurred in this work.

### 2.2. Systematic mapping studies and systematic literature reviews

Systematic Mapping Studies (SMS) and Systematic Literature Reviews (SLR) are secondary studies whose aim is to underpin the state-of-art about a subject. These methodologies aim at obtaining knowledge about the literature. However, there is a difference between these two methodologies. SLR aims to carry out an in-depth analysis of research questions which empirical studies can investigate. The SMS seeks to present an overview of the literature of an emerging research area. Therefore, SMS tends to be more descriptive, classifying the literature studies according to specific dimensions for answering general questions, such as what is being used in a research area, research methods, and publication fora are typical research questions of SMSs. Moreover, Systematic Mapping Studies have a broader scope about the SLR studies because it addresses several aspects of an emerging research field. Thus, graphs and summaries of data in a table are characteristics present in these studies. Furthermore, Systematic Mapping Studies provide a variety of benefits for both researchers and practitioners. These benefits are highlighted below.

**Benefits for researchers.** Researchers will have a guide for their future research and a list of primary studies that are state-of-art on cognitive load in software engineering. The results of the classification provide the focus of the selected studies. It was possible because of each research question points in which class studies are concentrated. This concentration of studies are points that researchers can explore in a more detailed SLR study. This study also provides a discussion containing the primary studies' contributions and also suggests future challenges. This study also adopted a well-defined protocol that is reproducible and paves the way for researchers to contribute and extend this study.

**Benefits for the software industry.** Despite SMS studies are well known for providing few contributions to practitioners, even so, we can highlight some of them. First, practitioners can benefit from a list of technologies related to the measurement of cognitive load, including sensors, machine learning techniques used, and metrics that can be a starting point for them setting-up future applications in the software industry. Second, this also anticipates to industry stakeholders a list of cognitive load related technologies that they could use to combine, test, and apply in software development environments. Third, researchers expect that cognitive load in software engineering to be beneficial in the long term for the software industry, for instance, providing stable techniques that classify code quality concerns before developers committing and inserting it into the source code.

## 2.3. Related works

This section describes works that are associated with this research. Table 1 summarizes the related works according to their title and reference, research method, number of studies categorized, their research aims, search protocol, search range, the dimensions investigated, and results. Moreover, this work assigned an identifier ID (RW) to each related work. This research describes the related work below.

**RW01**. Blasco et al. [22] reviewed the literature about wearable biometric recognition systems. Blasco et al. [22] covered articles published up until 2016. Blasco et al. [22] delimited the research domain to biometric recognition systems. The scope of this study was restricted to wearable sensors, such as wristbands. The authors aimed to provide a review of recognition systems that adopted wearable biometrics. Blasco et al. [22] categorized these systems according to the types of sensors, and the machine learning techniques applied to process the signals and signals measures. The results pointed out that researchers can apply multimodal measures to leverage classifiers' precision that deals with cognitive load measures. It is a relevant finding for engineers and researchers in software engineering to improve reconnaissance systems' security. This study is limited to the domain of biometric recognition, and the addressed dimensions of sensors and machine learning techniques did cover cognitive load issues.

**RW02**. Figl [27] introduced a literature review about the comprehensibility of business process models. The authors analyzed 79 articles to conduct a literature review. Figl [27] analyzed articles published until 2016. The Figl [27] research focuses on business process model comprehension. Figl [27] classified and discussed many aspects of business process model comprehension. The authors classified aspects such as the variables associated with model-process comprehension, situations that impact on the cognitive load increase during model comprehension, research outlets, tasks, and participants' characteristics. In software engineering, conceptual models are artifacts that could leverage developers' comprehension of the system features. Figl [27] concluded that cognitive load measures from psychophysiological sensors could lead to a better understanding of the cognitive process underlying the process model comprehension. The study of Figl [27] is focused on the comprehensibility of process models and discussed tasks and participants concerning this artifact. Our study has a broader scope of cognitive load expanding it to the software engineering. Furthermore, we addressed more dimensions such as machine learning techniques, purposes, sensors and metrics.

**RW03**. Gui et al. [13] reviewed studies which apply metrics based on users' brain for identification and authentication purposes. The authors surveyed about 188 articles to discuss several dimensions of EEG brain waves in recognition systems. Gui et al. [13] focuses its research on the domain of biometric recognition systems. Their study domain is focused mainly on brain-centric metrics due to their specific identification characteristics. They discussed dimensions such as the EEG devices commonly used to collect brain signals, a list of public EEG data, and machine learning classification techniques for EEG data. Authors found that Neural Networks (NN) trained with EEG brain waves classify with high correction recognition rate (CRR), usually between 60% and 90%. The study of Gui et al. [13] is strict to the domain of biometric recognition, and the addressed dimensions such as tasks, datasets, and classification techniques did not concern the users' cognitive load. In contrast, this systematic mapping classifies the datasets and machine learning techniques used in conjoint with cognitive load measures for application purposes in software engineering.

**RW04**. Bablani et al. [14] analyzed studies that focus on Brain–Computer Interfaces applied to the interdisciplinary context. The authors analyzed about 188 articles to respond to issues that concerned with signal acquisition and processing from BCI's. The analyzed articles cover studies published until 2018. Bablani et al. [14] focused mainly on the aspects of BCI devices, such as EEG, and machine learning techniques such as Support Vector Machine (SVM). Bablani et al. [14] aimed to make a general overview of BCI devices and machine learning techniques. For this, Bablani et al. [14] analyzed the articles they selected according to the categories of signal acquisition, the respective devices for each of them, the kind of brain signals, the measures works have extracted from the brain signals and a list of classification techniques. Authors found that studies applied various Machine learning classification algorithms to classify brain waves, such as Support Vector Machine, Neural Networks, and Linear Discriminant Analyses. Instead of concerning in a broader research field of Brain–Computer Interfaces, this mapping focus on cognitive load in software engineering.

**RW05**. Obaidellah et al. [28] reviewed the literature that focuses on the application of Eye-Trackers on empirical studies in software engineering. The authors analyzed 36 primary studies published between 1990 and 2014. This study is in the domain of program comprehension. Obaidellah et al. [28] aimed at providing an overview of how eye-tracking technology is handled in the software engineering domain to analyze program comprehension. Obaidellah et al. [28] investigated dimensions such as research topics, contributions, metrics, and devices involved in eye-tracking technology in software engineering. After analyzing the selected papers, Obaidellah et al. [28] found that most of the primary studies concerned about contributing to the program comprehension research topic. A great number of studies also adopted Java snippets as artifacts on experiments, adopted fixation and saccades metrics to analyze developers' cognitive load, and 41% of eye-tracking experiments conducted while developers executed comprehension tasks. In contrast, this systematic study is broader, consisting of the whole area of software engineering, reaching then on other types of sensors, such as EEG, fMRI, and fNIRS, and metrics related to cognitive load.

**RW06**. Sharafi et al. [29] surveyed the literature to analyze issues about the usage of Eye-Trackers technology on computer program activities. The authors selected 63 primary studies published between 1990 and 2016. The authors selected these studies aiming to provide key aspects of experiments involving eye tracking in computer programming in the domain of program comprehension. Sharafi et al. [29] classified selected primary studies according to dimensions such as tasks, artifacts, participants, eye-tracking devices, and metrics. The results point that 41% of papers conducted eye-tracking experiments on comprehension tasks, 38% papers adopted Java programming language as artifacts on experimental tasks, and 28% of papers considered the fixation count to analyze developers' performance on

**Table 1**
Summarization of related works.

| ID | Title | Research method | # Studies | Search protocol | Domain | Aim(s) | Period | Questions/Addressed dimensions | Results |
|---|---|---|---|---|---|---|---|---|---|
| RW01 | A Survey of Wearable Biometric Recognition Systems [22] | Survey | Does not Specify | No | Biometric Recognition | Explore specific issues that resides on wearable biometric recognition systems. | Until 2016 | Classification of wearable sensors, techniques, machine learning techniques, and discussion of research-domain issues such biometric datasets. | –Devices such as Apple Watch, Garmin Fenix 3 HR and Microsoft Band were used in studies;<br>–Classification machine learning techniques, such as, Bayesian Network, KNN, SVM, and Naïve Bayes were predominantly used with biometric signals;<br>–There is a lack of biometric recognition datasets; |
| RW02 | Comprehension of Procedural Visual Business Process Models A Literature Review [27] | Literature Review | 79 | Yes | Conceptual models comprehensibility | Categorize factors that influences cognitive load on comprehension of business process models. | Until 2016 | Presentation medium, model notations, tasks characteristics, and participants | –47% of the selected studies presented models representations on paper;<br>–Majority of studies focused on BPMN for representing process models;<br>–Experiments usually evaluated users on comprehension tasks;<br>–62% of primary studies recruited students to attend experiments; |
| RW03 | A Survey on Brain Biometrics [13] | Survey | ~188 | Yes | Biometric Recognition | Describe current scientific research on brain metrics intended to identification and authentication systems. | 2007 to 2017 | Tasks, datasets, and classification techniques. | –Tasks design in resting state protocol are the more common on BCI studies;<br>–Great part of studies analyzed and made assumptions based on the datasets available on Physionet;<br>−51 studies used KNN for brain waves classification on biometric recognition systems |
| RW04 | Survey on Brain–Computer Interface: An Emerging Computational Intelligence [14] | Survey | 181 | No | Brain–Computer Interfaces | Provide a survey about techniques involving ECOG, EEG, MEG, and MRI. | Until 2018 | Types of sensors, brain signals, techniques for feature extraction, and classification algorithms. | –EEG, fMRI, MEG, and fNRIS as main type of sensors to collect brain signals;<br>–ICA, Wavelet Transform, PCA and CSP as main techniques for extracting features;<br>–the SVM was the classifier which best performed on the data recorded for lie detection among NN, LDA and KNN; |
| RW05 | A Systematic Literature Review on the Usage of Eye-tracking in Software Engineering [29] | Literature Review | 36 | Yes | Program Comprehension | Provide an overview about the how eye-trackers are used in empirical studies, and highlight which contributions these studies provided to software engineering. | 1990 to 2014 | Research topics, contributions, metrics, types of Eye-Tracker devices. | 12 studies explored and contributed for the program comprehension research topic,<br>–16 studies adopted java as artifacts; Studies adopted fixation and saccades metrics; 47% of papers used Tobii Eye-Tracker. |
| RW06 | A Survey on the Usage of Eye-Tracking in Computer Programming [28] | Survey | 63 | Yes | Program Comprehension | Analyses key factors of experiments in computer programming which adopts Eye-Trackers; | 1990 to 2017 | Number of articles, tasks, artifacts, participants, type of eye-tracking devices, metrics. | –41% of papers conducted eye tracking experiments on comprehension tasks;<br>–38% papers adopted Java programming language as artifacts on experiments;<br>–44% of papers adopted Tobii Eye-Tracker;<br>–28% of papers measured the fixation count; |

comprehension tasks. This work did not treat the cognitive load as a central measure during program comprehension. In contrast, this Systematic Mapping Study focused on the developers cognitive load aspect, and therefore reached other types of sensors such as EEG, and heart beating, and tasks such as modeling and review tasks.

To sum up, we highlight that the differentials of this work is that it conducts a Systematic Mapping Study about the measures of cognitive load in the software engineering domain. The related works did not focused on cognitive load, but focused on specifically on program comprehension, or general research fields such as biometric recognition, and Brain–Computer Interfaces. For this, we adopted a well-defined search protocol, and found 63 primary studies published until May 2020. This classification addressed the dimensions of sensors, metrics, machine learning techniques, purposes, tasks, artifacts, participants, research methods, and research venues. Finally, the analysis contained in this study contributes to provide an overview about the state-of-the-art of the measures of cognitive load of developers, and pinpoint studies contributions and future challenges.

## 3. Methodology

This section presents the Systematic Mapping Study planning. For this, Section 3.1 introduces the objective and research questions explored. Section 3.2 describes the search strategy determined to retrieve a sample of potentially relevant works. Section 3.3 brings inclusion and exclusion criteria used to filter the retrieved works. Section 3.4 presents the study filtering process followed to find representative works. Section 3.5 describes the guidelines used for data extraction from selected articles. Section 3.6 presents the procedures used for data synthesis.

### *3.1. Objective and research questions*

This research aims to achieve two main objectives: (1) to classify primary studies about the measurement of developers' cognitive load; and (2) to pinpoint further challenges for future research. To reach these objectives the following research questions (RQ) explore and classify the primary factors and methods involved for measuring the cognitive load (RQ1–RQ8), and as well as the statistical aspects about the profile of this research field such as the research method (RQ9), and research venue (RQ10). This study investigates these aspects because they represent a general view of the process for measuring the cognitive load in software engineering. Moreover, the discussion section highlights further challenges about cognitive load measures in software engineering. Table 2 outlines the investigated RQs, along with their motivations and variables explored.

### *3.2. Search strategy*

The search strategy of this work comprises the conception of the search string, and the selection of a list of search engines to retrieve candidate studies.

#### *3.2.1. Conception of the Search String (SS)*

This section presents the guidelines to construct the search string, i.e., the terms used in search of search engines. The search string is composed of a set of mnemonics that delimits the scope, focus, and purpose of this systematic study. For this research, we identified the search terms using the PICo (Populations, Interventions, and Context), which is indicated to apply in qualitative studies [30,31]. This method has been used successfully in previous systematic review studies [19,32]. The literature preaches on dividing research questions into individual aspects [17,32]. These aspects form a list containing different keywords and synonyms, and combining these terms from this list using Boolean "ANDs" and "ORs" creates the search string. The PICo method consists of structuring the search string on the so-called Populations, Interventions, and Context. Concerning the original version of this study [20], we extended the search string with synonyms terms that we found in researches that measured cognitive load in software engineering.

The Populations part refers to the terms related to the standards and technologies. In this part, instead of maintaining the terms devices, sensors, and BCI we used in the previous work [20], we also included the terms EEG, ECG, EDA, and fMRI to the populations. These were terms we found in the literature that measure cognitive load in software engineering [3,10,23]. We also did not limit the population to software developers, as the studies carried on labs can recruit persons that are not developers to attribute it as a control group. The populations focused only on sensors because software engineering built up a new paradigm to measure the developers' cognitive load based on these sensors.

The Intervention is related to the terms that delineate the purpose of the previously mentioned technologies, i.e., the cognitive load measures in software engineering. Their synonyms, such as emotions, mental effort, and biometrics, were considered. Specifically, concerning the previous study, we placed the "mental effort" instead of the term "brain-synchronization", because this term was not found in software engineering previously [20]. We did not used component terms of cognitive load such as, stress, lack of sleep, and workplace pressures for avoiding restricting the search to these specific domains. We considered emotion as a synonym of the cognitive load due to two reasons: First, researchers in software engineering adopted the metrics to measure cognitive load to indicate negative and positive emotions [23,33]. Second, due to the evidence that emotions affect the cognitive load [34].

The Context segment of the search string contains the terms related to what research context in which practitioners and researchers expect contributions. Thus, this work considers terms inside the context of software engineering such as, diagram, code, bugs, software development, software testing, software maintenance, computer programming. In relation to the previous work [20], we added the synonyms program comprehension, software deployment, software modeling, and software analysis to the search string because these terms reinforce the context of the study in software engineering. Afterward, we connected each term with AND operator, as well as its synonyms with the OR operator.

In summary, we performed the following steps to define the search terms: (1) Define the primary terms (keywords) using PIO; (2) Identify synonyms, related or alternative terms to keywords; (3) Check if main articles on this research topic contain the selected keywords; (4) Associate synonyms, alternative words, or terms related to the primary keywords with the boolean "OR"; and, finally (5) Relate the major terms with boolean "AND". Table 3 shows three keywords and their synonyms.

The following Search String (SS) is the result of the combination of defined the terms and their synonyms. This search string was also that we applied to search engines.

*(devices OR sensors OR BCI OR EEG OR ECG OR EDA OR fMRI) AND ("cognitive load" OR emotions OR "mental effort" OR biometrics) AND ("software engineering" OR diagram OR code OR "software development" OR "software testing" OR "software maintenance" OR "computer programming" OR "program comprehension" OR "software deployment" OR "software analysis")*

#### *3.2.2. List of search engines*

Table 4 shows the list of search engines selected to retrieve potentially relevant articles. These electronic databases were chosen for two reasons. First, most of those search engines uncover works published in the main conferences, journals, and workshops in the research field of software engineering [35]. We also considered PubMED, a search

**Table 2**
Research questions investigated.

| Research questions | Objective | Variable |
|---|---|---|
| **RQ1:** How do the studies defined cognitive load? | The cognitive load seems a misused term in software engineering. This question maps the definitions of the term cognitive load in the software engineering research field. | Cognitive load |
| **RQ2:** What are the types of sensors adopted to measure the developers' cognitive load? | This question seeks to classify which sensors were already used in the software engineering research field to evidence what sensor studies used more, and what sensors studies neglected. | Sensors |
| **RQ3:** What metrics have been used to measure developers' cognitive load? | This question inquires to uncover which metrics selected studies related to cognitive load in software engineering. | Metrics |
| **RQ4:** What algorithms have been used to classify developer's cognitive load? | Reveal what ML techniques and which datasets were used by primary studies. | Machine learning |
| **RQ5:** For what purpose has developers' cognitive load been measured? | Map and underpin for which purposes academia concerned about addressing cognitive load in software engineering. | Purpose |
| **RQ6:** Which tasks have been used to measure developers' cognitive load? | Classify which tasks were considered essential in researches about cognitive load. | Software development tasks |
| **RQ7:** What were the artifacts used on cognitive tasks? | Discover which artifacts primary studies adopted to carry on experimental tasks to measure developers' cognitive load. | Artifacts |
| **RQ8:** How many participants did the studies recruit to measure developers' cognitive load? | Classify the studies according to a range of the number of participants, and describe its general distribution. | Number of participants |
| **RQ9:** Which research methods have been used to investigate cognitive load in software development tasks? | Disclose the research methods applied by studies about cognitive load in software engineering. | Research methods |
| **RQ10:** Where have the studies been published? | Uncover the kind of research venues used to publish the results about cognitive load in software engineering and analyze the recent trends of publication frequency. | Research venues |

**Table 3**
Definition of the main keywords and their synonyms that compose the search string.

| Major terms | Synonym terms |
|---|---|
| Sensors | Devices OR sensors OR BCI OR EEG OR ECG OR EDA OR fMRI |
| Cognitive load | Emotions OR "mental effort" OR biometrics |
| Software engineering | "software engineering" OR diagram OR code OR bugs OR "software development" OR "software testing" OR "software maintenance" OR "computer programming" OR "program comprehension" OR "software deployment" OR "software analysis" |

engine from the medical field, because we noted some papers [36,37] deepened their discussion on the neuroscience domain. Thus, other papers could also be published in research venues related to the medical context. Second, some previous studies [19,20] have shown their effectiveness in retrieving peer-reviewed articles for Systematic Mapping Studies.

### 3.3. Inclusion and exclusion criteria

We used the criteria defined in this section to include and exclude studies retrieved from search engines during the selection and filtering process. We used the Inclusion Criteria (IC) to add works to the sample of selected studies.

The Inclusion Criteria (IC) were used to include works for the selected studies sample. The IC included studies that:

- **IC1**: articles, book, or book chapters focused on approaches or algorithms for measuring the developer's cognitive load;
- **IC2**: publications in English;

**Table 4**
List of the search engines.

| Search engines | Link |
|---|---|
| ACM digital library | http://dl.acm.org/ |
| CiteSeerX library | http://citeseerx.ist.psu.edu/ |
| Google scholar | https://scholar.google.com.br/ |
| IEEE explore | http://ieeexplore.ieee.org/ |
| Inspec | http://digital-library.theiet.org/ |
| Microsoft academic | https://academic.microsoft.com/ |
| Pubmed | https://www.ncbi.nlm.nih.gov/pubmed/ |
| Scopus | http://www.scopus.com/ |
| Science direct | http://www.sciencedirect.com/ |
| Springer link | http://link.springer.com/ |
| Wiley online library | http://onlinelibrary.wiley.com/ |

- **IC3**: publications from journals, books, conferences, and workshops;
- **IC4**: publications made available up to May 2020, with no defined lower bound year.

Moreover, we used the Exclusion Criteria (EC) to filter out works from the selection process. The EC removed studies where:

- **EC1:** the title, abstract or any part of their content are not related to the research topic, but they are lexically associated with the terms of the search string;
- **EC2:** the study was not published in English;
- **EC3:** the authors removed definitions of patents, gray literature (non-peer-reviewed material), secondary studies, and talks descriptions;
- **EC4:** the abstract did not focus on the context of cognitive load in software engineering;
- **EC5:** the listed study appeared in duplicate; and
- **EC6:** the study did not cover issues that emerged from the defined research questions about measuring the developers' cognitive load in software engineering-related tasks.

### 3.4. Study filtering process

This section presents the process defined to filter works already published, which are closely related to our research questions (Section 3.1). This filtering process consists of eight steps and follows well-established guidelines recommended by the scientific community [16–18]. Recent literature mapping studies [20] have also demonstrated the effectiveness and usefulness of this adopted process. Thus, the process gradually filters potentially relevant studies to obtain a sample of representative studies. Fig. 1 introduces an overview of the filtering process, along with the results obtained in each step performed. This research got the list of representative studies of the literature after an initial group of 4175 works goes over complementary filtering steps, where the process selected 63 primary studies, all listed in Appendix. Each step of the filtering process is described as follows:

- **Step 1: Initial search.** The objective of this step is to gather the search engines' initial search results using the search string defined in Section 3.2.1 and no filters. In total, we gathered 4175 studies from eleven digital libraries.
- **Step 2: Remove impurities.** This step aims to remove irrelevant results from the initial search list, so we applied the exclusion criteria EC1 and EC2. Therefore, we removed calls for journal's special issues, calls for a conference paper, descriptions of talks, research reports, patents' specifications, and studies that have not undergone peer review. Thus, we removed 865 studies, i.e., 20.7% were filtered out, and then 3310 studies were made to the next step.
- **Step 3: Filter by title and abstract.** The objective of this step is to filter studies considering the title and abstract. By applying EC3 in this step, we removed studies without any relation to the research questions of this SMS. Thus, 95.2% of the studies were filtered out, and then 134 studies made to the next step.
- **Step 4: Combination.** This step aims to bring together the results from the previous step into a single directory to prepare for the next step. We did not apply any exclusion criteria in this step.
- **Step 5: Duplicates removal.** This step seeks to ensure the uniqueness of each study in our sample as a particular study can be retrieved from two or more search engines. We applied EC5 to remove all studies in duplicate. We removed 20.1% of the studies (27 of 134) in this step, remaining 107 unique studies toward the next step.
- **Step 6: Filtering studies by reading the full text.** The objective of this step is to filter studies considering their body texts. Thus, we applied EC6 over 107 papers and removed 57 studies that had made through up to this step, but their content did not show a relation with our SMS, then resulting in a set of 50 studies for the next step.
- **Step 7: Snowballing.** Concerning the previous study, we add this step to increase the coverage of the selection process by combining the list of studies obtained from the search string with a snowballing strategy [16,17]. Therefore, the selected papers in the last step were used as a starting set for iterating over their references, a process known as "snowballing" [38]. This process was iterated over two directions, i.e., backward (references which were listed inside a study) and forward (external references which cite the analyzed article). In each interaction, the articles were filtered by metadata (Title and Research Venue) applying EC1 and EC2, Abstract (EC4), and Full-text (EC6). The exact numbers are described in Fig. 2. Through this process, we found 17 new articles. These articles were included in the selection process, resulting in 67 candidate studies.
- **Step 8: Selection of representative work.** This step aims to provide additional reading and discussion step to make sure we were ending up the selection process without multiple versions (e.g., various papers) of the same study. Examining the 67 studies that remained in the last step, we identified four studies representations with the same underlying data. We then removed previous versions of those studies and maintained their recent version, resulting in the final set of 63 primary studies listed in Appendix.

### 3.5. Data extraction

This section describes the data extraction strategy used to answer the research questions defined in Table 2. Table 5 presents the data that the authors sought when performing data extraction in the selected studies concerning each research question. A form was prepared in excel so that the data in Table 5 could be collected. The table with the extracted data of all research questions is available on the website dedicated to this study.[1]

The authors defined the possible data that should be extracted from selected articles after a discussion. Two authors conducted a pilot extraction in a sample of 5 selected studies to confirm whether authors established a common understanding among the defined terms. Both authors classified 55 categories related to the ten research questions and the contributions category in these first five selected primary studies. We measured the Fleiss Kappa coefficient to measure the inter-rater agreement between these 55 pilot classifications authors both evaluated [39]. In these five selected studies, both authors agreed upon 48 classifications but disagreed about seven. The Fleiss Kappa coefficient returned a value of 0.792, which, according to Landis and Koch [40] indicates a substantial agreement between authors. The authors made a meeting to reason about the disagreements and reach a commonsense. During this pilot extraction, authors resolved all different interpretations related to the data classification that emerged between authors.

After this stage, the authors independently performed a data extraction on the remaining articles. After finishing this data extraction, the authors verified each other's results to validate and confirm the collected data.

### 3.6. Data synthesis

This section presents the methods used for synthesizing the data that authors extracted from the selected studies. Data synthesis consists of resuming the data authors obtained to summarize the results of this study, specifically, for each research question described in Section 3.1. It is challenging to conduct a meta-analysis on selected papers due to the studies' lack of homogeneity. We relied upon descriptive and frequency analysis and grounded theory to synthesize the answers of the defined research questions.

[1] https://luciangoncales.github.io/studies/SMSCognitiveLoad/.

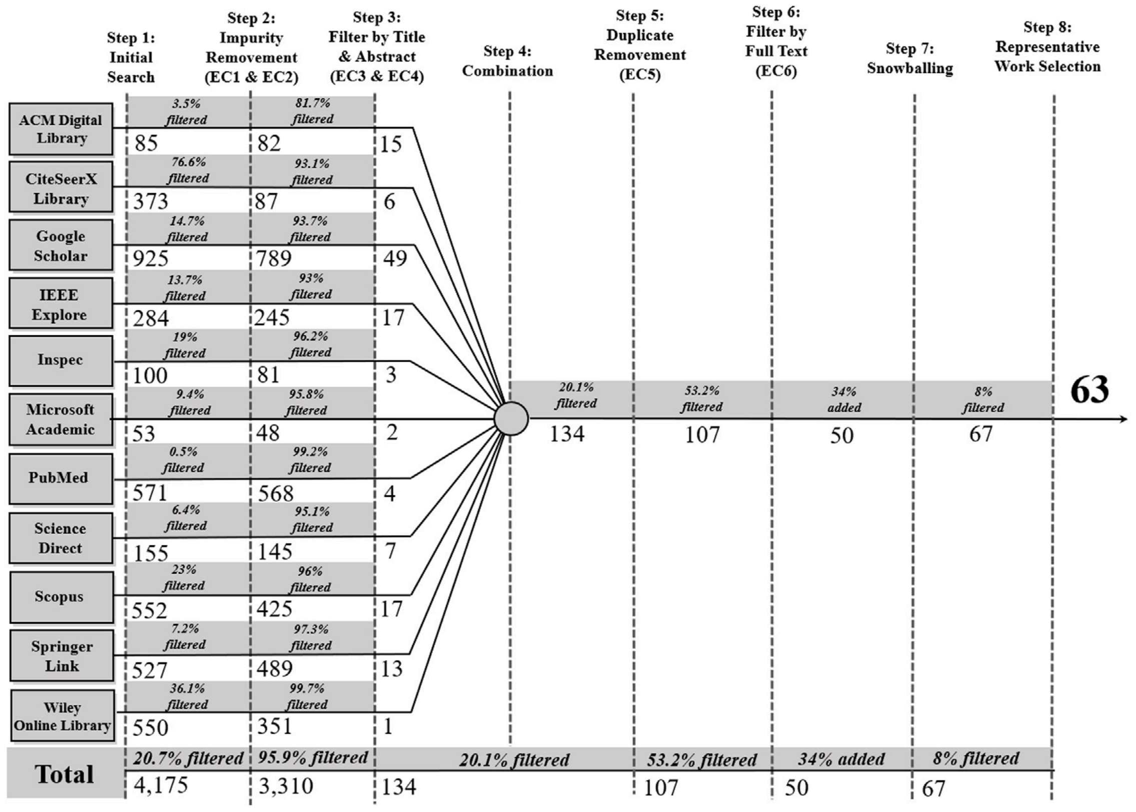

Step 1: Initial Search
Step 2: Impurity Removement (EC1 & EC2)
Step 3: Filter by Title & Abstract (EC3 & EC4)
Step 4: Combination
Step 5: Duplicate Removement (EC5)
Step 6: Filter by Full Text (EC6)
Step 7: Snowballing
Step 8: Representative Work Selection
ACM Digital Library 85 3.5% filtered 82 81.7% filtered 15
CiteSeerX Library 373 76.6% filtered 87 93.1% filtered 6
Google Scholar 925 14.7% filtered 789 93.7% filtered 49
IEEE Explore 284 13.7% filtered 245 93% filtered 17
Inspec 100 19% filtered 81 96.2% filtered 3
Microsoft Academic 53 9.4% filtered 48 95.8% filtered 2
PubMed 571 0.5% filtered 568 99.2% filtered 4
Science Direct 155 6.4% filtered 145 95.1% filtered 7
Scopus 552 23% filtered 425 96% filtered 17
Springer Link 527 7.2% filtered 489 97.3% filtered 13
Wiley Online Library 550 36.1% filtered 351 99.7% filtered 1
20.1% filtered 134
53.2% filtered 107
34% added 50
8% filtered 67
63
Total 4,175 20.7% filtered 3,310 95.9% filtered 134 20.1% filtered 107 53.2% filtered 50 34% added 67 8% filtered

**Fig. 1.** The final results relation according to each step of the filtering process.

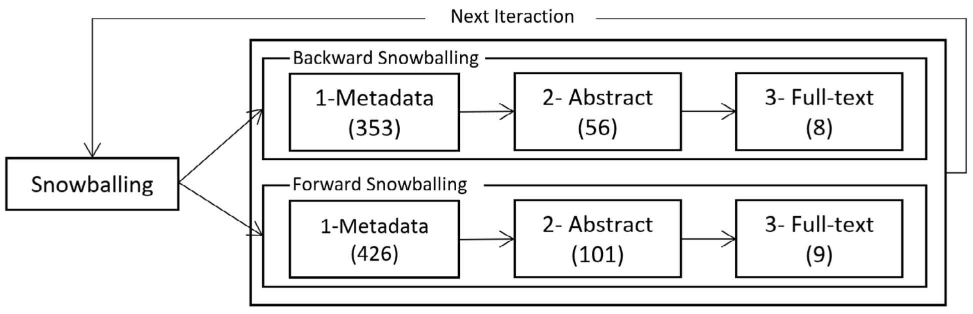


**Fig. 2.** The steps followed in the snowballing process.

We adopted a coding technique from Ground Theory to code the terms used in this study. This theory is generally composed of three steps for coding data, i.e., open coding, selective coding, and theoretical coding. Open coding consists of describing the process and behaviors succinctly from texts based on the author's perception. Thus, based on this data, the selective coding step identifies key categories for those descriptions, which denotes the data representativeness. We did not follow a full Grounded Theory approach since we did not build a theory from the collected data, thus not using the theory coding. Furthermore, this work has pre-existing research questions that delimited our analysis. The data analysis consists of a process that author first codes and then categorizes the data [41]. The coding step refers to extracting general issues from data succinctly, to turn this data able for labeling and thus classifying it. The categorization step contrasts the previously classified data to comprehend the variations between authors' perceptions. Based on these steps, right after reading the selected studies, the first two authors annotated their perceptions according to each research question's issues on a short memo. Thus, these notes were summarized and organized according to the classification scheme presented in Fig. 5 in Section 4.11.

The analyses of the results obtained from data extraction were made through the usage of spreadsheets. We used MS Excel to summarize the results based on the extracted data. The frequency analysis was done in all research questions. From research questions RQ1 to RQ9, we used tables to summarize the frequency and the distribution of selected papers for each answer. Furthermore, some selected studies could address more than a sensor (RQ2), or a machine learning technique (RQ4), and authors classified those studies as multimodal, and multi-algorithm, respectively. Therefore, we sought to provide fine-grained answers for cases like this, e.g., tabulating every sensor, and every machine learning technique used in these studies. In RQ7, besides the table containing the study classification and frequencies about software artifacts, we tabulated the software languages. In RQ10, a graph presents the research venue, year, and the number of studies produced per year. Thus, in the discussion section, a bubble chart shows the frequency of studies according to their contributions and the results of two research questions: purposes (RQ5) and research methods (RQ8). From this chart, we derived further challenges about cognitive load in software engineering.

**Table 5**
Data that were extracted from selected studies.

| Data | Description | Research questions |
|---|---|---|
| Cognitive load | The definition of cognitive load in software engineering such as the mental effort, or cognitive load theory. | RQ1 |
| Sensor | The sensor which studies used to collect data related to the cognitive load, such as the EEG, or fMRI. | RQ2 |
| Metric | The metric used to quantify or indicate the cognitive load spent by participants, such as the Event-Related Desynchronization (ERD), or pupil size | RQ3 |
| Machine learning technique | The machine learning techniques primary studies applied cognitive load measures, such as Support Vector Machine (SVM), and Neural Networks. | RQ4 |
| Purpose | There are various purposes in which the cognitive load was measured, such as code comprehension and task difficulty. | RQ5 |
| Tasks | There are several software engineering tasks in which researchers can measure the cognitive load, such as change or programming tasks. | RQ6 |
| Artifacts | The artifacts which developers or participants manipulated during cognitive load measurements, e.g., source code. | RQ7 |
| Number of participants | The number of participants that researchers recruited to attend the study, the data was collected in a range of 10 participants, e.g., 10–20, or 20–30 participants. | RQ8 |
| Research method | The research method of the study, e.g., controlled experiment, or evaluation study | RQ9 |
| Study type | The type of the study, e.g, conference, journal, workshop, or chapter. | RQ10 |
| Research venue | Extract the name of the research venue in which authors published the primary study. | RQ10 |
| Year | Obtain the year in which the study was published. | RQ10 |

**Table 6**
Classification of definitions primary studies used to reference cognitive load.

| Definitions | # Studies | % | Primary studies |
|---|---|---|---|
| Mental effort | 20 | 32% | [S01] [S02] [S05] [S06] [S11] [S15] [S20] [S22] [S25] [S29] [S31] [S34] [S38] [S46] [S47] [S56] [S59] [S60] [S61] [S63] |
| Cognitive performance | 6 | 10% | [S07] [S09] [S12] [S24] [S55] [S58] |
| Cognitive load theory | 4 | 6% | [S04] [S13] [S14] [S33] [S36] |
| Mental workload | 4 | 6% | [S48] [S50] [S51] [S52] |
| Cognitive load | 2 | 3% | [S03] [S16] |
| Does not define | 26 | 41% | [S08] [S10] [S17] [S18] [S19] [S21] [S23] [S26] [S27] [S28] [S30] [S32] [S35] [S37] [S39] [S40] [S41] [S42] [S43] [S44] [S45] [S49] [S53] [S54] [S57] [S62] |
| Total | 63 | 100% | |

## 4. Results

This section presents the results of the classification of the selected studies. Thus, the next section comprises on results of the research questions defined in Section 3.1.

### 4.1. RQ1: How do the papers define cognitive load?

RQ1 seeks to classify how primary studies defined cognitive load in software engineering. This research question is a new contribution to this research about the previous version [20]. Cognitive load is a term that researchers in software engineering misused. Therefore, this section looks for classifying the definitions of cognitive load in software engineering. Table 6 shows the classifications of the primary studies concerning the cognitive load definition.

A considerable part of primary studies (32%, 20/63) in software engineering defines cognitive load as a generic synonym of mental effort users spend when performing tasks. This definition does not relate the cognitive load to the cognitive load theory [42]. The cognitive load metrics indicators oscillate as users progressively spend more mental effort when solving a task. These studies do not present a preference for a specific sensor or metric to measure cognitive load.

Some primary studies (10%, 6/63) linked cognitive load to the cognitive performance term. The cognitive neuroscience research field use this term for referring to measure of how well users are processing cognitive tasks [2] in terms of their interplay between short and long term memory. Researchers measured the cognitive performance based on EEG frequency, mainly the theta and alpha waves.

A group of primary studies (6%, 4/63) defines cognitive load term from the cognitive load theory [42]. This theory defines three types of cognitive load: intrinsic load, extraneous load, and germane load. A task causes an intrinsic load on users due to the inherent complexity of the material. A task imposes an extraneous load on users when the material has a bad design. The acquisition of a new mental schema

**Table 7**
Classification of cognitive load types from cognitive load theory.

| Primary studies | Intrinsic | Extraneous | Germane | No specified |
|---|---|---|---|---|
| Crk. and Kluthe 2016 [S04] | ✓ | | | |
| Petrusel, Mendling, and Reijers 2017 [S13] | ✓ | | | |
| Tallon et al. 2019 [S14] | ✓ | ✓ | | |
| Lee et al. 2017 [S33] | | | | ✓ |
| Uysal 2013 [S36] | ✓ | ✓ | ✓ | |

imposes a germane load on users. Table 7 shows to what degree studies measured the cognitive load in software engineering. Crk and Kluthe 2016 [S04] and Petrusel, Mendling, and Reijers 2017 [S13] measured the intrinsic cognitive load during the experiments. Authors cited that the frequency bands and eye fixation oscillated according the artifact complexity. Kluthe 2016 [S04] specified that he removed components that could cause extraneous cognitive on participants. Tallon et al. 2019 [S14] affirmed that external features on the process model imposed extraneous load on participants. Lee et al. 2017 [S33] not specified which cognitive load they were measuring, differently from Uysal 2013 [S36] that measured three cognitive load types. However, studies have not followed the framework proposed by Sweller [1], i.e., the studies did not intend to reduce extraneous cognitive load for improving users learning.

Some primary studies (6%, 4/63) refers to cognitive load as the mental workload. These studies used the mental workload term as the amount of load a task can impose in the user's short-memory specifically. The primary studies in this category did not cite studies of cognitive load theory, neither cite the literature researchers used in cognitive neuroscience [2]. These studies used evidence that indicators such as oxygenate hemoglobin rises as the cognitive load grows.

About 3% (2/63) of the primary studies defined the cognitive load as the amount of load that a task imposes specifically to the users working memory capacity [43]. The researches in this category have the characteristic of using mainly eye blinks measures.

The majority of primary studies used the cognitive load term, but they did not define or related it to a specific theory. We found three types of measurements in this category. The first type [S26] [S40] [S43] [S45] [S54] [S57] measured the blood oxygen level to identify areas of the brain related to the source code. The second pattern of primary studies [S19] [S32] [S39] measured the cognitive load with blood volume pulse, heart rate, EEG frequency bands, and electrodermal activity to indicate when developers should not be interrupted. The third and last pattern of primary studies [S41] [S44] [S62] measured the variations of all frequency bands from EEG to point to the developer's mental effort on the programming task. Finally, the concentration of the primary studies in the group that did not define the term cognitive load theory in Table 6 suggests that studies that measure cognitive load in software engineering tend to misuse this term.

### *4.2. RQ2: What are the types of sensors utilized to measure developers' cognitive load?*

RQ2 seeks to understand which sensor researchers have applied to measure developers' cognitive load. It can help other researchers and engineers choose which sensors in their future studies or applications. Table 8 presents the classification of technologies for measuring the cognitive load of developers. Concerning the previous study [20], Table 8 lists new categories of sensors that researchers used to measure cognitive load, i.e., the fNIRS and EMG.

Table 8 shows that the results changed about the previous study [20]. A great number of studies (43%, 27/63) preferred to use a multimodal approach to measure cognitive load, i.e., to make utilization of multiple sensors. The researchers preferred EEG in the original research [20], evidencing a switch of focus from EEG to multimodal sensors (Table 9).

Primary studies tend to adopt multimodal sensors for two main reasons. First, there is a wide range of sensors related to cognitive load at a low cost, e.g., EEG, skin, and breath sensors. Second, the need to strengthen the correlation of results with the study purpose. For instance, some studies combined sensors aiming to join temporal and spatial precision characteristics. Other studies captured data from different parts of the body.

Table 8 also shows that primary studies barely used sensors in a separated fashion for measuring the cognitive load. Studies in software engineering produced little empirical knowledge about each sensor for capturing data related to cognitive load. In particular, the list in Table 8 did not contain any research focused on adopting the electro-dermal, heart rate, and respiration rate sensors to measure the developers' cognitive load independently. Instead, studies adopted a set of sensors together as shown in Table 9. Researchers need to explore more and evaluate sensors separately to enable the software engineering area to produce low-cost systems for monitoring data related to developers' cognitive load.

Differently from the previous research [20], Table 8 shows that researchers adopted the electroencephalogram (EEG) as the primary sensors less frequently than the multimodal sensors. Whereas the adoption of the Eye-Tracker increased in relation to the previous study (only 2), and now are very close to the EEG. About 22% (13/63) of the studies in software engineering explored EEG to measure cognitive load. It is a low number because researchers in the Brain–Computer Interfaces (BCI) apply EEG more constantly in their research [44]. This sensor is relevant because cognitive electrophysiology strongly links the EEG readings direct to neural population-level interactions [45]. However, researchers may have given little attention to it because EEG tends to be hard to deploy in the users' heads and is very sensitive to external noises, such as interferences of light bulbs and cellphones [46]. Studies also little focused on using Eye-Tracker sensors (16%,10/63). The Eye-Tracker can track the eye-measures and the location in the software artifact in which users spend their cognitive load. The low adoption of the Eye-Tracker contributes to the lack of knowledge of how cognitive load is related to smaller units of software artifacts, such as the syntax or line of a source code.

Moreover, 13% (8/63) of the primary studies adopted the fMRI, and about 5% (3/63) adopted the fNIRS. The number of studies adopting the fMRI is also low because of the elevated cost of the equipment. The fMRI also needs a large space for the equipment. An alternative to these shortcomings and limitations is the fNIRS sensor, which provides similar information compared to the fMRI, but with lower spatial precision. The fNIRS equipment does not cover brain activity in deeper regions of the brain, which fMRI does. Even with the cost and mobility solved, only 3 three works adopted fNIRS. Devices which has spatial precision depend on the response of the blood flow in the brain causing a delay in the response. Therefore, researchers generally prefers devices with higher temporal precision.

Only 2% (1/63) adopted an EMG device to measure cognitive load. This device captures the electrical signals of muscle' movements in the face. The studies neglected the idea of using some patterns in facial muscles to indicate developers' cognitive load. Facial muscles also reflex the words users are thinking, a process called sub-vocalization.

Table 9 presents the works that used multimodal sensors. In the horizontal direction, Table 9 shows which sensors the primary studies used to measure cognitive load. Reading vertically, Table 9 shows what studies adopted a specific type of sensor. In relation to the previous study [20], Table 9 now enables to evidence the sensors studies not combined. Table 9 lists 27 studies that applied multimodal sensors, whereas the original research listed 12. Concerning the previous study, this table also includes the EMG, plethysmograph, and movement sensor.

**Table 8**
Classification of sensors for measuring the cognitive load of developers.

| Sensors | # Studies | % | Primary studies |
|---|---|---|---|
| Multimodal sensors | 27 | 43% | [S03] [S05] [S06] [S07] [S08] [S10] [S12] [S15] [S19] [S20] [S25] [S29] [S30] [S31] [S32] [S33] [S34] [S35] [S37] [S38] [S39] [S40] [S46] [S56] [S59] [S61] [S63] |
| EEG | 13 | 22% | [S04] [S09] [S17] [S23] [S24] [S41] [S44] [S51] [S52] [S55] [S58] [S60] [S62] |
| Eye-Tracker | 10 | 16% | [S02] [S11] [S13] [S14] [S18] [S21] [S27] [S28] [S42] [S47] |
| fMRI | 8 | 13% | [S01] [S22] [S26] [S43] [S45] [S49] [S54] [S57] |
| fNIRS | 3 | 5% | [S36] [S48] [S50] |
| EMG | 1 | 2% | [S53] |
| Does not use | 1 | 2% | [S16] |
| Total | 63 | 100% | |

**Table 9**
Classification of primary studies that used multiple sensors.

| Primary studies | Set of combined sensors | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | EEG | Eye-tracker | GSR | EDA | EMG | fNIRS | fMRI | ECG | RR | HR | PPG | Movement |
| Nargess Nourbakhsh 2013 [S03] | | ✓ | ✓ | | | | | | | | | |
| Sarah Fakhoury 2018 [S05] | | ✓ | | | | ✓ | | | | | | |
| Thomas Fritz 2016 [S06] | ✓ | ✓ | | ✓ | | | | | | ✓ | | |
| Magdalena Borys 2017 [S07] | ✓ | ✓ | | | | | | | | | | |
| Sebastian C. Müller 2015 [S08] | ✓ | ✓ | | ✓ | | | | | | ✓ | | |
| Daniela Girardi 2017 [S10] | ✓ | ✓ | | ✓ | | | | | | | ✓ | |
| Randall Minas 2017 [S12] | ✓ | | | ✓ | ✓ | | | | | | | |
| Norman Peitek 2019 [S15] | | ✓ | | | | | ✓ | | | | | |
| Manuela Züger 2015 [S19] | ✓ | ✓ | | ✓ | | | | | | | | |
| Norman Peitek 2018 [S20] | | ✓ | | | | | ✓ | | | | | |
| Davide Fucci 2019 [S25] | ✓ | | | ✓ | | | | | | ✓ | | |
| Sebastian C. Müller 2016 [S29] | | | | ✓ | | | | | ✓ | ✓ | | |
| Sebastian C. Müller 2015 [S30] | ✓ | ✓ | | ✓ | | | | | | | | |
| Norman Peitek 2018 [S31] | | ✓ | | | | | ✓ | | | | | |
| Manuela Züger 2018 [S32] | ✓ | | | ✓ | | | | | | ✓ | | |
| Seolhwa Lee 2017 [S33] | ✓ | ✓ | | | | | | | | | | |
| Ricardo Couceiro 2019 [S34] | ✓ | ✓ | | ✓ | | | | ✓ | | | | |
| Toyomi Ishida 2019 [S35] | ✓ | ✓ | | | | | | | | | | |
| Thomas Fritz 2014 [S37] | ✓ | ✓ | | ✓ | | | | | | | | |
| Florian Schaule 2018 [S38] | | | ✓ | | | | | | | ✓ | | |
| Manuela Züger 2018 [S39] | | | | | | | | ✓ | | ✓ | | ✓ |
| Yu Huang 2019 [S40] | | | | | | ✓ | ✓ | | | | | |
| Norman Peitek 2018 [S46] | | ✓ | | | | | ✓ | | | | | |
| Sarah Fakhoury 2020 [S56] | | ✓ | | | | ✓ | | | | | | |
| Ricardo Couceiro 2019 [S59] | | ✓ | | ✓ | | | | ✓ | | | | |
| Ricardo Couceiro 2019 [S61] | | ✓ | | | | | | | | ✓ | | |
| Devjeet Roy 2020 [S63] | | ✓ | | | | | ✓ | | | | | |

**Legend:**
EEG: Electroencephalogram, GSR: Galvanic Skin Response, EDA: Electro Dermal Activity, EMG: Electromyography,
fNIRS: Functional near-infrared spectroscopy, fMRI: Functional Magnetic Resonance Imaging,
ECG: Electrocardiography, RR: Respiration Rate, HR: Heart Rate, PPG: Plethysmograph.

Table 9 shows primary studies combined the EEG, Eye-Tracker, and EDA sensors more frequently. These sensors were also that primary studies often combined. Combining these two sensors enables the analysis of brain waves and tracking where the developer's eye is looking at the software artifact. The primary studies also used frequently the heart rate sensors which also has a high correlation with cognitive load. The blank spaces evidences that mainly few studies have combined information from fMRI or fNIRS with a sensor that has greater temporal precision, such as, the Eye-Tracker, or the EEG. The lack of this combination evidence a gap in studies to relate the brain dynamics that involve the cognitive load in specific locations of software artifacts in more detail. For this, researchers in software engineering could explore a neuroscience research field dedicated to the conjoint analysis of EEG and fMRI data called EEG-fMRI [47]. The lack of exploration of GSR and ECG sensors indicates that primary are missing more precise aspects from heart beatings and skin measurements.

Table 9 also shows that primary studies adopted the plethysmograph and motion sensors rarely. But the lacking of these sensors in primary studies does not impact the results. The heart beating sensors can capture breathing data. The EEG and Eye-Tracker sensors can detect movements that interest researchers in software engineering, i.e., eye blinks and eye saccades.

### *4.3. RQ3: What metrics have been used to measure developers' cognitive load?*

RQ3 classifies which metrics the primary studies used to measure developers' cognitive load. We intend to provide an overview of indicators that researchers have used to measure the developers' cognitive load. Table 10 presents the classification of primary studies according to the measures of cognitive load. Table 10 lists a new category of metrics that researchers used to measure cognitive load, i.e., the

**Table 10**
Classification of primary studies based on metrics related to cognitive load.

| Metrics | # Studies | % | Primary studies |
|---|---|---|---|
| Multimodal measurements | 39 | 62% | [S02] [S03] [S05] [S06] [S07] [S08] [S10] [S11] [S13] [S14] [S17] [S19] [S20] [S21] [S23] [S24] [S25] [S27] [S28] [S29] [S30] [S31] [S32] [S33] [S34] [S35] [S37] [S38] [S39] [S40] [S42] [S45] [S46] [S48] [S56] [S57] [S58] [S59] [S61] |
| Blood oxygenation levels | 8 | 13% | [S01] [S22] [S26] [S36] [S43] [S49] [S50] [S54] |
| Frequency bands | 6 | 10% | [S41] [S51] [S52] [S53] [S60] [S62] |
| Event Related Desynchronization (ERD) | 4 | 7% | [S04] [S12] [S44] [S55] |
| Eye fixation | 2 | 3% | [S18] [S47] |
| Individual Alpha Frequency (IAF) | 1 | 2% | [S09] |
| Does not specify | 3 | 5% | [S15] [S16] [S63] |
| Total | 63 | 100% | |

blood oxygenation level, and studies that did not specify the metrics. Concerning the previous version [20], we removed some metrics of this table, such as the Event-Related Potential (ERP), Event-Related Spectral Perturbation, and Steady-State Visual Evoked Potential (SSVE). The studies did not apply the ERP, SSVE metrics in software engineering. Therefore, the new classification presents a more precise outlook about the metrics in which primary studies had adopted [20].

The main finding is that most works (62%, 39/63) use multimodal measurements to indicate developers' cognitive load levels. In this work, papers that used or applied more than one indicator were classified as "multimodal measurements". Researches in the recognition systems field [14,22] also usually use multimodal biometrics to get over of inconsistencies during data analysis. Thus, failure caused by readings in one of the sensors would be covered by another. Even so, a great number of primary studies (35%, 22/63) tends to use only one metric to measure developers' cognitive load. Most of these indicators researchers extracted from brain activity, i.e., blood oxygenation levels (13%, 8/63). Researchers usually extracted these metrics from fMRI or fNIRS sensors. Moreover, 19% (11/63) of metrics were obtained from an electroencephalogram, such as frequency bands (10%, 6/63), Event-Related Desynchronization (7%, 4/63), and Individual-Alpha Frequency (2%, 1/63).

Researchers used the blood oxygenation levels to estimate the source related to a specific brain area that correlates with software engineering tasks [10,26]. The blood oxygen level is a consolidated metric to indicate the brain activation areas. In the original research [20], this metric was not present, and its emerging adoption evidences a rising interest of researchers concerning the developers' brain activity level to cognitive load.

Researchers calculated the Event-Related Desynchronization (ERD) to measure mental effort during source code comprehension [3,4]. This measure is useful to capture the more frequent activity of brain waves during the comprehension event. In an example of the application of frequency bands, Kosti et al. [12] demonstrated that a statistically significant difference in frequency bands could evidence the task difficulty level of software developers. In general, the presence of metrics such as the frequency bands, ERD, and IAF individually in primary studies shows that researchers have a common understanding that brain waves can measure cognitive load, but a divergence on what aspect of the brain wave should be considered to measure cognitive load. Researches in neurophysiology related these brainwave metrics to users' cognitive load level in activities that required mental workload from them, such as the chess game.

Only two studies focused on tracking eye fixations from developers. The eye fixation represents where developers are looking at the software artifact after some time. The number of studies focusing exclusively on this metric implies an insufficient body of knowledge about the developers' points of interest in the source code. However, researchers in software engineering also focused less on eye fixation because they preferred combining eye-tracking metrics with other metrics, such as EEG. Finally, the primary studies that did not define metrics consists on studies in the initial research phase.

Table 11 presents the works that used multimodal metrics. In the horizontal direction, Table 11 depicts the metrics that primary studies used to measure cognitive load. Reading vertically, Table 11 lists the primary studies which apply the metric the column indicates. Overall, the studies do not have a preferred metric for measuring cognitive load. In relation to the previous study [20], Table 11 lists 38 studies that applied multimodal sensors, whereas the original research listed 13. The previous research listed 20 different metrics in Table 11 [20]. In the current work, we were able to identify 40 metrics.

There are only a few metrics that primary studies adopted frequently. The primary studies preferred to use all frequency bands (AFB) from the EEG sensor (12/38). In the Eye-Tracker, the studies adopted the number of eye blinks (9/38). Primary studies also preferred to measure electrodermal activity (EDA) from the metrics that capture skin activity. Although we identified four metrics from an fNIRS sensor, the studies chose two metrics, the HbO (Deoxygenated Hemoglobin) and the HbR (Oxygenated Hemoglobin). The Blood Volume Pulse metric obtained through fMRI appears in three primary studies. The studies preferred using the metric heart rate variability (11/38) from the sensors that capture cardiac signals.

Reading horizontally, Table 11 presents a positive and a negative aspect regarding the combination of metrics primary studies combined. The positive side is that there is a varied range of metrics concerning each type of sensor. Thus, from a sensor, researchers have several options for measuring cognitive load. The negative aspect is that the blank gaps indicate that the studies did not seek to use all the metrics each sensor provides. Thus, it is evident that the studies that preferred to combine metrics could explore more, but they chose to restrict the collection to some metrics for each sensor. It makes the correlation in the collected data viable during an analysis. For example, on the Eye-Tracker, while the reduced frequency of blinking eyes indicates high cognitive load demand, researchers also can use the reduced pupil size to confirm this high task demand.

### *4.4. RQ4: What algorithms have been used to classify developers' cognitive load?*

RQ4 seeks to classify the different machine learning techniques that the studies applied. Table 12 presents a list of mapped machine learning algorithms and the corresponding primary studies. Concerning the previous study [20], Table 12 does not contains the categories of K-Means, Logistic Regression, Neural Network, Relevance Vector Machine, and Linear Regression.

Table 12 shows that 30% (18/63) of primary studies generally adopted classification techniques. About 11% (7/68) of primary studies evaluated multi-algorithms of classification. The results changed significantly from the previous study [20]. Primary studies were inclined to adopt Naïve Bayes (NB) or the Support Vector Machine (SVM) in the original research [20]. However, they turned the focus on evaluation multi-algorithms of classification.

Table 12 shows that Naïve Bayes (NB) and Support Vector Machines (SVM) were the most used machine learning techniques among

**Table 11**
Classification of primary studies based on multimodal metrics.

| Studies | EEG | | | | | | | | | | | Eye-tracker | | | | | | | | | | | Skin | | | | | fNIRS | | | | fMRI | | | Heart | | | | | PA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AFb | FBR | Ab | Bb | Tb | Gb | ERD | PS | AR | AL | ML | EB | EF | NF | NS | SD | FD | GPL | BN | BR | PDS | VI | AG | PSG | ST | SCR | EDA | HbT | HbO | HbR | Oxy | BOL | I | V | HR | HRV | BVP | RR | SL | TPE |
| Michael Zimoch 2012 [S02] | | | | | | | | | | | | | | ✓ | ✓ | | | ✓ | | | | | | | | | | | | | | | | | | | | | | |
| Nargess Nourbakhsh 2013 [S03] | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | ✓ | ✓ | | | | | | | | | | | | | | | | |
| Sarah Fakhoury 2018 [S05] | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | ✓ | ✓ | | | | | | | | | |
| Thomas Fritz 2016 [S06] | ✓ | | | | | | | | | | | ✓ | ✓ | | | | | | | | ✓ | | | | | | ✓ | | | | | | | | ✓ | ✓ | ✓ | ✓ | | |
| Magdalena Borys 2017 [S07] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Sebastian C. Müller 2015 [S08] | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | ✓ | | | | | | | | | | | | | | |
| Daniela Girardi 2018 [S10] | ✓ | | | | | | | | | | | ✓ | ✓ | | | | | | | | ✓ | | | | | | ✓ | | | | | | | | ✓ | ✓ | ✓ | | | |
| Razvan Petrusel 2016 [S11] | | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | | | | | | | | | | | | |
| Razvan Petrusel 2017 [S13] | | | | | | | | | | | | | | ✓ | | | ✓ | | | | | | | | | | | | | | | | | | | | | | | |
| Miles Tallon 2019 [S14] | | | | | | | | | | | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | |
| Seolhwa Lee 2016 [S17] | | | ✓ | ✓ | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Manuela Züger 2015 [S19] | ✓ | | | | | | | | | | | | | | | | | | | | | | | | ✓ | | | | | | | | | | ✓ | | | | | |
| Mahnaz Behroozi 2018 [S21] | | | | | | | | | | | | ✓ | | | ✓ | | | | | | | ✓ | | | | | | | | | | | | | | | | | | |
| Jefferson Seide Molléri 2019 [S23] | | | | | | | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Aruna Duraisingam 2017 [S24] | ✓ | ✓ | | | | | ✓ | | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Davide Fucci 2019 [S25] | ✓ | | | | | | | | | | | | | | | | | | | | | | | | ✓ | | | | | | | | | | ✓ | ✓ | | | | |
| Katja Kevic 2016 [S27] | | | | | | | | | | | | ✓ | ✓ | | | ✓ | | | | | | | | | | | | | | | | | | | | | | | | |
| Martin Konopka 2015 [S28] | | | | | | | | | | | | | ✓ | | | | | | | | ✓ | | | | | | | | | | | | | | | | | | | |
| Sebastian C. Müller 2016 [S29] | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | | ✓ | | | | | | | | ✓ | ✓ | | ✓ | | |
| Sebastian C. Müller 2015 [S30] | ✓ | | | | | | | | | | | ✓ | ✓ | | | | | | | | ✓ | | | | | | ✓ | | | | | | | | ✓ | ✓ | ✓ | ✓ | | |
| Norman Peitek 2018 [S31] | | | | | | | | | | | | ✓ | | | | | | | | | ✓ | | | | | | | | | | | ✓ | | | | | | | | |
| Manuela Züger 2018 [S32] | ✓ | | | | | | | | | | | ✓ | | | | | | | | | | | | | | | ✓ | | | | | | | | ✓ | ✓ | | | ✓ | |
| Seolhwa Lee 2017 [S33] | ✓ | | | | | | | | | | | ✓ | | | | | | | | | ✓ | | | | | | | | | | | | | | | | | | | |
| Ricardo Couceiro 2019 [S34] | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | |
| Toyomi Ishida 2019 [S35] | | | ✓ | | | | | | | | | | | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Thomas Fritz 2014 [S37] | ✓ | | | | | | | | | | | ✓ | | | | | | | | | | | | | | | ✓ | | | | | | | | | | | | | |
| Florian Schaule 2016 [S38] | | | | | | | | | | | | | | | | | | | | | | | ✓ | | ✓ | | | | | | | | | | ✓ | ✓ | | | | |
| Manuela Züger 2018 [S39] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | ✓ | ✓ |
| Yu Huang 2019 [S40] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | ✓ | | | | | | | | |
| Mahnaz Behroozi 2018 [S42] | | | | | | | | | | | | | | | ✓ | | | ✓ | | | | ✓ | | | | | | | | | | | | | | | | | | |
| Benjamin Floyd 2017 [S45] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | | | | |
| Norman Peitek 2018 [S46] | | | | | | | | | | | | | | ✓ | ✓ | | | | | | | | | | | | | | | | | ✓ | | | | | | | | |
| Yoshiharu Ikutani 2014 [S48] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | | | | | | | | |
| Sarah Fakhoury 2019 [S56] | | | | | | | | | | | | | | ✓ | | | | | | | | | | | | | | ✓ | ✓ | ✓ | ✓ | | | | | | | | | |
| Ryan Krueger 2020 [S57] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ✓ | ✓ | | | | | | |
| Ramaswamy Palaniappan 2019 [S58] | | | | | | | ✓ | ✓ | ✓ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Ricardo Couceiro 2019 [S59] | ✓ | | | | | | | | | | | | | | | | | | | | ✓ | | | | | ✓ | | | | | | | | | | ✓ | | | | |
| Ricardo Couceiro 2019 [S61] | ✓ | | | | | | | | | | | | | | | | | | | | ✓ | | | | | | | | | | | | | | | ✓ | | | | |

**Legend:**
AFb: All Frequency bands, FBR: Frequency band ratio, Ab: Alpha, Bb: Betha, Tb: Theta, Gb: Gamma, ERD: Event related Desynchronization, AR: Asymmetry Ratio, AL: Attention level, ML: Meditation level, EB: Eye-blink, EF: Eye-fixation, NF: Number of fixations, NS: Number of saccades, PSS: Pupil size, SD: Saccades duration, NS: Number of Saccades, FD: Fixation Duration, BN: Blink Number, BR: Blink Rate, GPL: Gaze Path Length, VI: Visual Intake, AG: Accumulative GSR, PSG: Power spectrum GSR, ST: Skin Temperature, SCR: Skin Conductance Response, EA: Electrodermal Activity, HbT: Total Hemoglobin, HbO: Deoxygenated Hemoglobin, HbR: Oxygenated Hemoglobin, Oxy: Total Oxygenation Changes, BOL: Blood oxygen levels, I: Images, V: Voxels, HR: Heart rate, HRV: Heart Rate Variability, BVP: Blood Volume Pulse, RR: Respiratory Rate, SL: Sleep level, TPE: Time spent on physical activity, PA: Physical Activity.

the papers that applied a single technique (15%, 9/63). In relation to the original research [20], the adoption of the Naïve Bayes is higher than the Support Vector Machine. The studies which adopted Naïve Bayes conducted comparisons with other machine learning techniques such as Support Vector Machines (SVM) and Decision Trees (DT). The authors chose the Naïve Bayes classifications because it overcame the performance concerning the SVM and DT. In this way, the authors preferred reporting the Naïve Bayes results over the SVM classifier and the DT. However, this is somewhat a limitation of the research in cognitive load in software engineering. Future works should include and report the effectiveness and classifications of the Support Vector Machine because it is robust to the curse of dimensionality issue. The adoption of Naïve Bayes and Support Vector Machine are in concordance with results presented by Gui et al. [13], Blasco et al. [22].

A minority of primary studies adopted other classification algorithms such as Decision Tree (2%, 1/63), and Random Forest (2%, 1/63), summing up to 4% (2/63). Random Forest is another technique that primary studies neglected and deals well with the curse of dimensionality. Few primary studies (6%, 4/63) did not specify a machine learning technique because they are solution proposals that intend to use an algorithmic machine learning computation, but it is was not specified yet.

Another finding is that machine learning techniques such as the K-Nearest Neighbor (KNN) and Neural Networks (NN) do not appear in Table 12 because these techniques are prone to overfitting with psychophysiological data. Literature in BCI research points to these techniques to be susceptible to overfitting problems with EEG data primarily [44]. Because these limitations, authors adopted KNN and neural networks to compare respective efficacy precisely with SVM and Naïve Bayes in Table 12.

Table 13 lists the double of studies in relation to the previous version [20]. In the first version, studies usually combined 5 different machine learning techniques [20], while in this study Table 13 identified 11 different techniques.

Reading vertically, Table 13 shows which primary studies contain a specific machine learning technique. Thus, the Support Vector Machine, Naïve Bayes, and Random Forest (RF) techniques were the techniques that the primary studies most combined. As previously mentioned, SVM and RF are robust techniques for a small amount of data, which justifies the authors' preference for these techniques. Table 13 shows that possible to note that only a few studies have evaluated the performance of the Decision Tree, Linear Discriminant Analysis, Linear Regression, Gradient Boosting, and Quadratic Discriminant Analysis. These machine learning techniques have a limited potential for working effectively in conjoint with psychophysiological data. However, modifications and improvements in the algorithms can make these techniques more useful. For example, we did not notice in the primary studies the adoption of the adaptive machine learning techniques. This technique enables the update of the classification parameters according to the incremental arrival of new data. Another example is the modified version of the LDA, i.e., the sLDA which is more robust to smaller data, however primary studies have not explored it [44].

In the horizontal direction, Table 13 also shows that only one primary study combined the classifiers. Ensemble classifiers are relevant in this research field. Researchers missed the opportunity to explore more the potentials of ensemble classifiers instead of assessing their effectiveness individually.

**Machine learning datasets:** Table 14 presents the description of the datasets used in the primary studies, such as the purpose of primary studies using the dataset, the type of data contained on the dataset, the number of participants, the number of tasks and their duration, the filters applied on collected data, and the format. Primary studies generated datasets for key applications in software engineering such as cognitive load classification, task difficulty, emotion classification, work interruption, and the detection of source code quality concerns. We did not discuss the primary studies datasets and machine learning techniques' performance in the original version [20].

These datasets generally contain rich data usually combined by a significant number of participants and multimodal data such as the EDA, EEG, and Heart Rate Variability data, as found in Fucci et al. [48] [S25]. Muller and Fritz [23] [S34] formed a dataset that contains HR, HRV, RR and EDA. Muller and Fritz [23] collected these data from 10 participants for classifying the presence of source code quality concerns. Although the number of participants was limited, this dataset has a considerable size because the authors monitored participants during a whole day of work. Despite the high value of these data sets, they are not available online due to the confidentiality of participants' data.

**Performance:** Reporting which of the machine learning techniques performed best is a difficult task due to three main reasons. First, there is a difference in the sizes of datasets. Second, researchers do not have a consensus on what performance indicators to adopt. For instance, some works present only the sensitivity; others measure both precision and recall. Third, because these works trained the machine learning techniques in different conditions, some techniques adopt cross-validation of 10 folders, and others use 5. Therefore, unified analysis of these techniques lacks in the literature to conclude the performance of these techniques.

Therefore, this work summarizes some of the best results obtained through the datasets presented in Table 14.

- **Cognitive load classification:** A KNN machine learning reached 90.4% of accuracy to classify cognitive load based only on Eye tracking trained with five-fold cross-validation [S07];
- **Emotion:** A tree classifier J48 trained with frequency band ratios, heart rate variability and pupil size achieved 64.32% of precision and 82.03% of recall to classify negative and positive emotions trained in a leave-one-out method [S08];
- **Level of expertise:** An SVM trained with EEG + Eye-Tracker data achieved a precision of 97.7%, a recall of 96.4% and f-measure of 97% to classify between two class of expertise level (Novice/Expert) on 10-fold cross-validation by a leave-one-out approach [S33];
- **Program comprehension:** A multi-layer perceptron reached 95% of precision and 90% recall to identify if a participant is dealing with a code or a prose task based on heart signal trained with hold-out cross-validation [S25];
- **Source code quality concerns:** A Random Forest classifier trained with HR, HRV, RR, and EDA achieved 13% of precision and 38.6% of recall to identify the presence or not of five categories of quality concerns [S29];
- **Task difficulty:** A Naïve Bayes approach reached a precision of 90,1%, and a recall of 73,4% for classifying two levels of difficulty (easy/difficult) [S24];
- **Work interruption:** The accuracy of a Naïve Bayes reached 91.5% on lab study, while 78.6% on the field to classify the propensity for interruption or not [S19].

### 4.5. RQ5: For what purpose have researchers measured developers' cognitive load?

RQ5 tries to reveal for what purpose researchers have measured developers' cognitive load. Table 15 presents the classification of primary studies according to the studies' purposes.

Concerning the previous study [20], instead of showing a tendency, Table 15 confirms that primary studies have an established preference for the cognitive load on code comprehension purposes. The majority of the primary studies (38%, 24/63) were concerned with analyzing developers' cognitive load to analyze code comprehension. The focus on code comprehension presented in Table 15 is a fact which the literature

**Table 12**
Algorithms and approaches for measuring the cognitive load of developers.

| Machine learning algorithms | # Studies | % | Primary studies |
|---|---|---|---|
| Multi-algorithms of classification | 7 | 11% | [S03] [S07] [S21] [S25] [S38] [S39] [S58] |
| Naïve Bayes (NB) | 6 | 10% | [S06] [S19] [S24] [S30] [S32] [S37] |
| Support Vector Machines (SVM) | 3 | 5% | [S23] [S33] [S34] |
| Decision Tree (DT) | 1 | 2% | [S08] |
| Random Forest (RF) | 1 | 2% | [S29] |
| Does not use | 41 | 65% | [S01] [S02] [S04] [S05] [S09] [S11] [S12] [S13] [S14] [S16] [S17] [S18] [S20] [S22] [S26] [S27] [S31] [S35] [S36] [S40] [S41] [S42] [S43] [S44] [S45] [S46] [S47] [S48] [S49] [S50] [S51] [S52] [S53] [S54] [S55] [S56] [S57] [S59] [S60] [S61] [S62] |
| Does not specify | 4 | 6% | [S10] [S15] [S28] [S63] |
| Total | 63 | 100% | |

**Table 13**
List of primary studies that applied more than one machine learning technique.

| Primary Studies | Set of machine learning techniques | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | SVM | NB | DT | LDA | LR | KNN | NN | RF | GB | QDA | EC |
| Nargess Nourbakhsh 2013 [S03] | ✓ | ✓ | | | | | | | | | |
| Magdalena Borys 2017 [S07] | ✓ | | ✓ | ✓ | ✓ | ✓ | | | | | ✓ |
| Mahnaz Behroozi 2018 [S21] | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | | |
| Davide Fucci 2019 [S25] | ✓ | ✓ | | | | ✓ | ✓ | ✓ | | | |
| Florian Schaule 2018 [S38] | ✓ | ✓ | | | | | | ✓ | | | |
| Manuela Züger [S39] | ✓ | ✓ | | | | | ✓ | ✓ | ✓ | | |
| Ramaswamy Palaniappan [S58] | ✓ | ✓ | ✓ | | | ✓ | ✓ | ✓ | | ✓ | |

**Legend:**
SVM: Support Vector Machine, NB: Naïve Bayes, DT: Decision Tree, LDA: Linear Discriminant Analysis,
LR: Logistic Regression, KNN: K-Nearest Neighbor, EC: Ensemble Classifier, RF: Random Forest,
GB: Gradient Boosting, QDA: Quadratic Discriminant Analysis.

of eye-tracking on software engineering also identified [28,29]. Program comprehension is an elementary activity in software engineering. The main concern of these studies was investigating which brain areas were more or less involved in programming comprehension. Moreover, code comprehension is a cognitive process that has support for the traditional theory of program comprehension [49]. It is central to software maintenance tasks.

Concerning the original research [20], the interest of primary studies in investigating task difficulty increased, representing a higher amount than task difficulty. Primary studies (19%, 12/63) aimed to identify the difficulty level of tasks based on developers' cognitive load measurements. Identifying the task difficulty level for developers can be an essential indicator in software engineering activities, such as allocating human resources for different tasks, or avoiding bug insertion [23]. Researchers could also use the task difficulty to predict the completion time for a task [7].

Moreover, 10% (6/63) of primary studies aimed to measure developers' cognitive load to identify work interruption. Some studies (3%, 2/63) investigated the developers' reactions in terms of emotions during software-development activities, and investigated the users' stress levels. A minority of primary studies focused on using the cognitive load data to evaluate productivity (5%, 3/63). The work interruption is a relevant factor toward improving developer productivity. The interruption of a concentrated developer in a task impairs his resumption and performance. Researchers must focus more on these purposes to avoid developers spending more cognitive load to complete their assignment. Primary studies also focused little on the developers' mental state and emotions. This lack contributes to limit the knowledge of how stress and negative emotions impact users' cognitive load. These investigations can be crucial to improving productivity and how the lack of developers' well-being affects the code quality.

Furthermore, the primary studies also investigated cognitive load with the purpose of tracking code dependencies (2%, 1/63), estimate code complexity (2%, 1/63), and programming learning (2%, 1/63). Furthermore, about 6% (4/63) of primary studies focused on identifying when software developers are prone to insert quality concerns on source code, such as bugs. Some primary studies (5%, 3/63) measured developers' level of expertise by using brain wave activity [3, 4]. The original research [20] does not contain primary studies concerned with investigating cognitive load for identifying the level of expertise, code complexity, identification of code dependencies, and programming learning. The low focus of primary studies in these classes reinforced that primary studies barely investigate the cognitive load in an application domain.

Few studies investigated quality problems in the source code, pointed the developer productivity, and measured code complexity based on developers' cognitive load. These researches have the potential to transform the software industry. The software industry currently relies on traditional software metrics, such as cognitive or cyclomatic complexity, which calculate the number of cycles a source code performs. These metrics, along with a wide range of software metrics, have little correlation with code comprehension [11], which compromises the accuracy of the software development indicators. Developers use these metrics to point out code quality problems, estimate developers' delivery times, and refactoring when the code's complexity is too high. Therefore, there is a chance that these activities will suffer from high inaccuracy and thus jeopardize the software development process while depending only on metrics based on source code. Cognitive load, being a developer-based and individual metric, potentially could calculate these tasks more accurately. For example, a high cognitive load could point to a code that developers should slice into smaller pieces and turning it easier to maintain.

Only 2% (1/63) of primary studies focused on measuring cognitive load to support learning programming. Learning support based on feedback from students' cognitive load would enable building more effective learning programming materials. The lack of support for students learning to program is a factor that could contribute to the formation of qualified professionals. The little development in these applied areas indicates a gap in providing valuable contributions to a real development scenario.

**Table 14**
List of datasets produced in primary studies.

| Study | Type | Data | Participants | Number of Tasks/Duration | Filters | Format |
|---|---|---|---|---|---|---|
| [S03] | Cognitive load classification. | Galvanic skin response recorded at 10 Hz, eye data, i.e., blink number recorded at 50 Hz | 13 | 8 arithmetic tasks | Does not specify | Text |
| [S07] | Cognitive load Classification. | EEG Signal recorded in a Mitsar EEG 201 with 21-channels sampling rate of 500 Hz, Eye data recorded at 50 Hz with ETG2.0 | 20 | 6 arithmetic tasks, 5,5 s each, one second of resting between tasks. | Movements artifacts removal with PCA on EEG data. | Text |
| [S08] [S06] | Emotion classification | EEG brain waves obtained from a single-channel at 512 Hz. Electro-dermal activity (EDA), skin temperature, heart rate, blood volume pulse (BVP), and pupil size from the Eye-Tracker Eye Tribe. | 17 | 2 programming tasks, 30 min each task. There is an interval between the two tasks, that is a two-minute video showing a fish (Relax task). | Low-pass, and a high-pass filter, butterworth filter applied on EDA signal for obtain phasic and tonic part. | Text |
| [S19] [S06] | Work interruption | EEG signal obtained from a Neurosky recorded at 512 Hz, this dataset frequency bands, EDA, and Skin Conductance Level collected from a wrist band, eye measurements obtained from an Eye-Tracker. | 10 | A two-hour session on a continuous programming task. | 50 Hz notch filter on EEG Brain waves. Butterworth filter from extracting eye measures, | Text |
| [S21] | Stress levels | Saccade and blink eyes data obtained from SMI head-mounted at 60 Hz | 11 | 2 programming tasks conducted at a whiteboard, and another conducted on papers/No task limit reported. | Not applied filters | Text |
| [S24] | Task difficulty | EEG signal captured with Emotiv Epoc headset, which has 14-channel, recorded at 128 Hz. In this dataset contains the, energy of different bands, ERD, Frequency ratios, Asymmetry Ratio. | 8 | 20 code comprehension questions, relax state of 30 s between tasks. | Elliptic IIR filter, band-pass filtering from 0.5–3 Hz | Text |
| [S25] | Program comprehen-sion | EEG signals from 1-channel device (512 Hz), EDA (4 Hz), and BVP (64 Hz) from Empatica E4. | 28 | 3 sessions containing 3 code comprehension (60 s) and 6 prose (30 s) comprehension tasks. | Band-pass filter applied on EEG and BVP data. | Text |
| [S29] | Source code quality concerns | HR, HRV and RR data and EDA obtained with wristband and Sense core chest strap. Source code metrics such as McCabe Complexity, and Halsteads Complexity | 10 | One continuous day of work during programming tasks. | Butterworth filters on EDA signal. | Text |
| [S30] [S06] | Task difficulty | EEG signal obtained from a Neurosky recorded at 512 Hz, this dataset frequency bands, EDA, and Skin Conductance Level collected from a wrist band, eye measurements obtained from an Eye-Tracker. | 15 | 8 short code comprehensions tasks | 50 Hz notch filter on EEG brain waves. Butterworth filter from extracting eye measures, Butterworth filters on EDA signal | Text |
| [S32] | Work interruption | EEG signal obtained at 512 Hz, Empatica E3 to capture EDA, and Skin temperature and Blood Volume Pulse, Heart Rate obtained from the Polar H7 sensor. | 33 | 10 developers conducted programming tasks for one hour; 10 participants conducted programming tasks for two hours; 13 participants conducted programming tasks for two weeks. All tasks developers suffered intermittent interruptions. | Did not specify | Text |

(continued on next page)

**Table 14** (*continued*).

| Study | Type | Data | Participants | Number of Tasks/Duration | Filters | Format |
|---|---|---|---|---|---|---|
| [S33] | Classify level of expertise. | EEG with a 32 channel iCap System which records a at a sample rate of 500 MHz, Eye-Tracker with eye position data. | 38 | 23 comprehending source code with duration about 60 s, relaxing task of 60 s for baseline. | Did not specify | Text |
| [S34] | Cognitive load classification. | Heart rate variability from an ECG. | 26 | 3 code comprehension tasks with time limit of 10 min, control activity of duration of 60 s. | Did not specify | Text |
| [S37] | Task difficulty. | EEG signal recorded at 512 Hz from 1-channel. EDA recoded at 8 Hz from Affectiva Q Sensor, Gaze data recorded at 300 Hz from TX300 eye-tracker | 15 | 8 code comprehension tasks. | DC component removed the EDA, low-pass Butterworth filter in EDA. Notch filter at 60 Hz in EEG. | Text |
| [S38] | Work interruption | Galvanic skin response, Skin temperature, Heart rate, Heart rate variability | 20 | 3 tasks which consist on remembering a square, and solving math equations, each task has a limit of 8 min. Participants were interrupted intermittently | Movements wrist artifacts were removed from data. | Text |
| [S39] | Work interruption. | HR and HRV data Polar H7 chest strap, HR, fisical activity, and sleep from Fitbit Charge 2. | 13 | Monitoring the participants daily tasks, Participants wore sensors during all their daily tasks. | Did not specify. | Text |
| [S58] | Task difficulty. | EEG signal from Emotiv Epoc+ with 14-channels, ERD, Asymmetric Ratio (ASR). | 9 | 6 comprehension tasks, which the completion time varied from 35 s to 210 s. | Elliptic IIR filters | Text |

**Table 15**
Classification of primary studies based on primary studies purpose.

| Purpose | # Studies | % | Primary studies |
|---|---|---|---|
| Code comprehension | 24 | 38% | [S01] [S02] [S05] [S09] [S11] [S13] [S14] [S15] [S17] [S18] [S20] [S22] [S35] [S43] [S44] [S45] [S46] [S47] [S48] [S52] [S54] [S56] [S57] [S63] |
| Task difficulty | 12 | 19% | [S03] [S06] [S24] [S26] [S27] [S30] [S31] [S37] [S40] [S41] [S53] [S58] |
| Work interruption | 6 | 10% | [S19] [S23] [S32] [S38] [S39] [S55] |
| Cognitive load classification | 5 | 8% | [S07] [S34] [S42] [S50] [S59] |
| Quality concerns | 4 | 6% | [S29] [S49] [S61] [S62] |
| Level of expertise | 3 | 5% | [S04] [S25] [S33] |
| Productivity | 3 | 5% | [S08] [S16] [S51] |
| Emotion recognition | 2 | 3% | [S10] [S12] |
| Code complexity | 1 | 2% | [S60] |
| Identify code dependencies | 1 | 2% | [S28] |
| Programming learning | 1 | 2% | [S36] |
| Stress levels | 1 | 2% | [S21] |
| Total | 63 | 100% | |

### 4.6. RQ6: Which tasks have been used to measure developers' cognitive load?

RQ6 investigates the tasks that researchers have used to measure the developers' cognitive load which are disclosed in Table 16.

The main finding is that most of the primary studies (83%, 52/63) analyzed cognitive load, mainly during programming tasks. Secondary studies also pointend that programming task is common in this research field [28,29]. In relation to the original research [20], the preference of primary studies for conducting programming tasks for measuring cognitive load increased significantly. It evidence that this task is a standard rather than a mere tendency.

Considering the previous work [20], we also deepened the classification and we found out that most primary studies followed different patterns of programming tasks, i.e., comprehension, and changing tasks in Table 17. Comprehension tasks is an activity which developers must observe and mentally process the source code instructions to deduce their output. In change tasks, developers modify research questions by extending and adding new lines to the source code. Table 17 presents the primary studies mapped according to these categories.

In relation to the previous study [20], Table 16 lists four new categories, including the tasks of modeling, review, data structures rotation, and memorization. The emerging of these classes shows that researchers in software engineering seek to analyze the developers' cognitive load in tasks that are closer to the software engineering context.

In particular, Table 16 also shows that a minority of works focuses on Modeling tasks (4%, 4/63) in which developers analyzed business process models. Researchers also evaluated the cognitive load of developers during review tasks (5%, 3/63). In some studies [S16] [S47], these tasks consisted of reviewing changes on the source code before being submitted to the repository. In a primary study [S23] the cognitive load of participants was measured during reviewing prose texts. About 3% (2/63) of primary studies adopted arithmetic equation tasks. Only 2% (1/63) of studies consisted of developers making mental tasks for rotating data structures. In particular, the tasks presented a 3D data structure to the developers, which should mentally

Table 16
Classification of primary studies according to tasks.

| Tasks | # Studies | % | Primary studies |
|---|---|---|---|
| Programming | 52 | 83% | [S01] [S04] [S05] [S06] [S08] [S09] [S10] [S12] [S15] [S17] [S18] [S19] [S20] [S21] [S22] [S24] [S25] [S26] [S27] [S28] [S29] [S30] [S31] [S32] [S33] [S34] [S35] [S36] [S37] [S39] [S41] [S42] [S43] [S44] [S45] [S46] [S48] [S49] [S50] [S51] [S52] [S53] [S54] [S55] [S56] [S57] [S58] [S59] [S60] [S61] [S62] [S63] |
| Modeling | 4 | 6% | [S02] [S11] [S13] [S14] |
| Review tasks | 3 | 5% | [S16] [S23] [S47] |
| Arithmetic equation calculation | 2 | 3% | [S03] [S07] |
| Data structures rotation | 1 | 2% | [S40] |
| Memorization and solve arithmetic equation | 1 | 2% | [S38] |
| Total | 63 | 100% | |

recognize an equivalent representation of this structure according to different options. Only 2% (1/63) of primary studies were required from participants to perform a task for memorizing a square position followed by a task to solve a simple equation. The purpose was to analyze how interruptions could jeopardize a low demanding cognitive task (memorization) compared to a median demanding cognitive task (arithmetic equation).

Despite the exploration of these categories, studies did not explore it enough. Actually, there is a few studies which focused on review activities, which is pivotal to ensure quality generally before commits to the repository, and modeling activities, which developers perform to model business processes and software architecture.

In addition to this limitation, the primary studies did not consider investigating other activities that do not appear in Table 16, i.e., conducting unit software tests and software deployments activities, such as developers executing the system configuration and deployment steps. Thus, there is little knowledge about the cognitive load with tasks related to the software process as a whole. The other aspect that the studies missed to investigate was the cognitive load of activities that are fundamental for programming. Researchers in software engineering have barely explored tasks such as solving arithmetic calculations, data structures, and memorizing artifacts. The programming task is a complex task that involves these fundamental tasks. It is an indication that software engineering studies are skipping relevant steps to relate cognitive load and the programming task theoretically. Therefore, it would be important that more studies seek to understand how the cognitive load of the developer interacts when solving arithmetic calculation, understanding the spatial arrangement of data structures, and memorization tasks. The sum of the cognitive load that developers spent on these tasks could produce a granular method for calculating the developers' cognitive load in programming tasks.

Overall, the primary studies focused more on programming tasks. Table 17 presents the primary studies mapped according to the categories previously described. As observed in this Table 17, majority of these programming tasks demand developers to comprehend source code. Only one study considered both kinds of tasks, i.e., comprehension and change tasks. The study of Müller [21] considered both these tasks because he first investigated the viability of classifying the task difficulty level during comprehension tasks. And after, in the next stage, Müller [21] investigated if it was possible to predict task difficulty while developers are changing the source code.

While there is a high concentration of studies that performed programming tasks, a much smaller number focused on code change tasks, i.e., extending and adding features to the source code tasks. The smaller number of studies on change tasks indicates the difficulty of implementing an experimental setup to evaluate this task. None of the studies use a stand-alone tool to integrate the data from sensors into the integrated development environment. Another factor is the difficulty of analysis since code extension tasks require hand movements. Users' movements increase the number of non-related artifacts in the psychophysiological data, i.e., noise.

Table 17
Classification of primary studies according to programming tasks.

| Programming tasks | Primary studies |
|---|---|
| Change tasks | [S08] [S10] [S12] [S18] [S19] [S21] [S24] [S27] [S32] [S39] [S42] [S53] [S55] [S57] |
| Comprehension tasks | [S01] [S04] [S05] [S06] [S09] [S15] [S17] [S20] [S22] [S25] [S26] [S28] [S29] [S31] [S33] [S34] [S35] [S36] [S37] [S41] [S43] [S44] [S45] [S46] [S48] [S49] [S50] [S51] [S52] [S54] [S56] [S58] [S59] [S60] [S61] [S62] [S63] |
| Comprehension and change tasks | [S30] |

## *4.7. RQ7: What were the artifacts used on tasks to measure cognitive load?*

RQ7 investigates which artifacts that researchers adopted to evaluate the cognitive load in the software engineering research field. Table 18 presents the classification of primary studies in relation to the artifacts. The main finding is that primary studies focused on adopting the source code as the chief artifact used to evaluate cognitive load in software engineering. Thus, 81% (51/63) of primary studies adopted source code in their experimental tasks. The main difference in relation to the previous study [20] is that these updated results presents that the source code is an established artifact for studies of cognitive load in software engineering, instead begin a merely trend. However, primary studies split their focus mainly in Java, C and C++ programming languages to conduct to measure developers cognitive load (Table 19).

We also found some studies that used Business Process Models (4%, 4/63) and math equations (3%, 2/63) as artifacts. A minority of studies adopted artifacts such as plain text (3%, 2/63), equations and polygons (2%, 1/63), data structures (2%, 1/63), source code, and plain text (2%, 1/63). Concerning the original research [20], we found two new categories, i.e., Business Process Models and data structures. The emerging of these categories indicates primary studies started to explore artifacts that are more closely related to software engineering. Despite this evolution, the artifacts that compose software development activities are diverse, e.g., different types of source code languages, textual documentation, and diagrams. The primary studies are far from cover the totality of variations of artifacts contained in software engineering projects.

For instance, 4% (4/63) of primary studies used different process models to measure cognitive load in empirical studies. A primary study applied various process models, such as Business Process Model Notation (BPMN), eGantt, EPC, and Petri Net [S02]. Some primary studies adopted only the BPMN diagram [S11] [S13] [S14].

Only one study added data structures as an artifact. The artifact, specifically, is a binary tree. This study also does not represent a series of existing data structures, such as AVL trees. As we observed in the last section, it is necessary to obtain knowledge of how these specific artifacts demand cognitive load from developers because programming is a too abstract task.

**Table 18**
Classification of primary studies based on artifacts.

| Artifacts | # Studies | % | Primary studies |
|---|---|---|---|
| Source code | 51 | 81% | [S01] [S04] [S05] [S06] [S08] [S09] [S10] [S12] [S15] [S16] [S17] [S18] [S19] [S20] [S21] [S22] [S24] [S25] [S26] [S27] [S28] [S29] [S30] [S31] [S32] [S33] [S34] [S35] [S36] [S37] [S39] [S41] [S42] [S43] [S44] [S46] [S48] [S49] [S50] [S51] [S52] [S53] [S54] [S55] [S56] [S58] [S59] [S60] [S61] [S62] [S63] |
| Process models | 4 | 6% | [S02] [S11] [S13] [S14] |
| Equation | 2 | 3% | [S03] [S07] |
| Plain text | 2 | 3% | [S23] [S47] |
| Source code & Natural language | 2 | 3% | [S45] [S57] |
| Data structures | 1 | 2% | [S40] |
| Equation & Polygons | 1 | 2% | [S38] |
| Total | 63 | 100% | |

**Table 19**
Classification of primary studies based on source code artifacts.

| Source code artifacts | Primary studies |
|---|---|
| Java | [S01] [S04] [S08] [S09] [S10] [S12] [S16] [S17] [S18] [S19] [S22] [S24] [S27] [S29] [S30] [S34] [S35] [S36] [S42] [S43] [S46] [S52] [S54] [S56] [S58] [S59] [S60] [S61] [S62] |
| Supported any source code | [S15] [S20] [S21] [S63] |
| C#: | [S06] [S37] |
| C\C++ programming language | [S40] [S44] [S49] [S50] |
| Natural language and C language | [S25] [S45] [S57] |
| Java and C++: | [S05] |
| C#, C++, and Java | [S28] |
| Pseudo-code and Java | [S26] |
| Pseudo-code | [S48] |
| Did not specify | [S31] [S32] [S33] [S51] [S53] [S55] |

Moreover, the number of two studies adopting equations is not representative of the vast number of mathematical equations and formulas within programming problems. The research focused on simple calculations such as sums and division, and they did not require knowledge outside the participants' knowledge domain. The software consists of several complex equations varying according to the application domain. For example, different accounting formulas composes an e-commerce management software and physics calculations in graphics engines. The developers' lack of knowledge in these math formulas implies a way to verify the developers' expertise. Therefore, the studies did not focus on equations focused on specific knowledge domains.

Only two studies covered textual content. The software engineering projects include specification documents such as project descriptions with software requirements, documents containing specifications of the developed software, and reference documentation of programming languages.

The artifacts were generally short containing few components, e.g., a binary tree with up to 5 nodes, a math equation with up to 3 operators, or source code with around ten lines. Researchers focused on controlling factors on artifacts that could turn the task more difficult, such as the presence or not of bugs on source code [37], and the presence or not of disrupted identifiers [15]. Authors argue that they usually must adapt the task's size to enable participants to complete the tasks.

Table 19 presents the type of source code artifacts these studies adopted. We can observe that the most adopted source code artifact among primary studies is Java. In general, researchers adopted Java because it is the most popular programming language in the developers' community. Also, because Java was the language in which participants were more used too. Previous studies [28,29] also found this pattern. Few studies [S40] [S45] analyzed the impact of different software artifacts, i.e., between prose (plain text) and source code in relation to cognitive load indicators. For instance, Floyd et al. [50] discovered that distinct brain regions were activated when developers interacted with code and pseudo-code artifacts.

The primary studies focused on adopting the java source code because participants were familiar with this language. The researchers' objective was to use Java to avoid participants spent any extra load with aspects of a language that they do not know. However, we believe that researchers would explore this aspect better they should adopt the pseudo-code. The reduced number of studies using the pseudo-code artifact may indicate that studies have not avoided participants spending an extra load on the unnecessary details of programming languages. Another observation is that researches do not reflect the reality of programming projects. Different programming languages that Table 19 does not shows usually compose a real software project. Studies did not even use artifacts such as PHP, javascript, MySQL, and various APIs such as angular and nodeJS. Hence, the primary studies did not cover the practices of real projects in the software industry integrally.

### *4.8. RQ8: How many participants did the studies recruit to measure developers' cognitive load?*

RQ8 seeks to investigate the number of participants in which studies recruited in their experiments for measuring the cognitive load in software engineering. To answer this question, we classified the studies on five different ranges of participants' number. Table 20 presents the classification of the primary studies according to the range of participants they recruited. Concerning the original research [20], Table 20 also lists works that recruited between 70 and 80 participants and studies that did not specify the number of participants.

More than half of the studies (53%, 33/63) recruited less than 20 participants to measure the cognitive load. Some studies (21%, 13/63) recruited less than 10 participants. A significant portion of studies (24%, 15/63) recruited between 21 and 30 participants to collect their cognitive load data. This portion of participants is the same found in early studies [28,29]. Researchers recruited between 31 and 40 participants in 8% (5/63) of the primary studies; 3% (2/63) of primary studies recruited between 41 and 50 participants. Finally, only 5% (3/63) recruited between 70 and 80 participants, which makes the research an exception concerning the number of participants.

Specifically, most studies focus on the range of 11 to 20 participants. Recruiting between 11 to 20 participants appears to be a limitation

**Table 20**
Classification of primary studies based on number of participants.

| Number of participants | # Studies | % | Primary studies |
|---|---|---|---|
| 0–10 | 13 | 21% | [S19] [S23] [S24] [S29] [S35] [S41] [S44] [S50] [S51] [S53] [S55] [S58] [S62] |
| 11–20 | 20 | 32% | [S01] [S03] [S05] [S06] [S07] [S08] [S17] [S21] [S22] [S26] [S30] [S36] [S37] [S38] [S39] [S42] [S43] [S47] [S49] [S52] |
| 21–30 | 15 | 24% | [S02] [S10] [S18] [S20] [S25] [S27] [S31] [S34] [S45] [S46] [S56] [S57] [S59] [S60] [S61] |
| 31–40 | 5 | 8% | [S04] [S09] [S14] [S32] [S33] |
| 41–50 | 2 | 3% | [S12] [S16] |
| 70–80 | 3 | 5% | [S11] [S13] [S40] |
| Does not specify | 5 | 8% | [S15] [S28] [S48] [S54] [S63] |
| Total | 63 | 100% | |

**Table 21**
Classification of primary studies based on research method.

| Research methods | # Studies | % | Primary studies |
|---|---|---|---|
| Validation studies | 51 | 83% | [S01] [S03] [S04] [S05] [S06] [S07] [S08] [S09] [S11] [S13] [S14] [S16] [S17] [S18] [S19] [S21] [S24] [S26] [S27] [S29] [S30] [S31] [S32] [S33] [S34] [S35] [S36] [S37] [S38] [S39] [S40] [S41] [S42] [S43] [S44] [S45] [S46] [S48] [S49] [S50] [S51] [S52] [S53] [S55] [S56] [S57] [S58] [S59] [S60] [S61] [S62] |
| Proposal of solution | 5 | 6% | [S15] [S20] [S28] [S54] [S63] |
| Replication study | 3 | 2% | [S22] [S25] [S47] |
| Experience paper | 2 | 3% | [S02] [S23] |
| Philosophical paper | 1 | 2% | [S12] |
| Replication proposal | 1 | 2% | [S10] |
| Total | 63 | 100% | |

of studies. However, a large number of participants is not required to analyze the results. Concerning the previous study [20], we found that the authors recruited more than 30 participants in 15 primary studies. Practically, the number of studies within the range of 21–30 participants decreased. On another side, the number of participants increased little in 41–50 and 70–80 categories. This reduced number evidence the difficulty of authors in recruiting participants for the experiments. A large number of participants is not mandatory, as long as it is sufficiently representative.

Concerning the previous study [20], we described the distribution of participants by task (Fig. 3). Fig. 3 presents the distribution of the number of participants per task (RQ6). The bold line represents the median in the middle of the blank square.

We found out that the number of participants recruited per study who performed tasks to change and comprehend code is very close to the general distribution (All). The median of all participants' distribution (All) is about 18 participants, and its 75th quartile is 30. This concentration means that in general, the number of participants researchers recruited resides between 20 and 30 participants. The distribution of participants in comprehension and change tasks follows almost the same pattern of the distribution of all participants. These tasks are the most representative in this research field. The boxplot of comprehension tasks shows that the number of participants is, in general, a bit lower than the number of participants of the change tasks. As the boxplots are upper skewed, this means that the distribution is not normal. The space between the 75th quartile and the maximum values evidences the few studies recruits between around 30 and 50 persons. Thus, studies that recruit more than fifty participants are evident outliers in this field of research.

Recruiting participants is a challenging task. First, the time for recruiting participants is limited because the deadlines for producing research results are short. Moreover, the study design can also restrict the selection of participants. For example, the number of participants would be lower if the experimental study's research objective be focused on developers with the desired skill, then discarding the others that do not have this specific skill, e.g., programming in Java Language. Furthermore, participants do not like to attend experiments to measure cognitive load [46]. These experiments require them to perform difficult programming tasks, and researchers may deploy a bunch of sensors throughout the participants' bodies. These deployment procedures are tedious and turn the participants to be afraid to attend [46].

### *4.9. RQ9: Which research methods have been used to investigate cognitive load in software development tasks?*

RQ9 classifies the primary studies according to research methods categories. These studies were classified based on the categories that Wieringa et al. [51] proposed. Wieringa et al. [51] suggested categorizing software engineering studies in evaluation studies, proposals of solution, philosophical paper, replication, and personal experience research methods. Table 21 presents the primary studies in relation to these categories. Table 21 lists four new classes of research methods about the original research [20], including replication studies, experience paper, philosophical paper, and replication proposal.

The majority of primary studies (81%, 51/63) were validation studies. Concerning the original research [20], we found out new classes of studies related to validation methods. These classes are experimental studies in the lab and experiment studies in the field. Table 22 lists the primary studies that are within these categories. This classification clarifies which studies researchers conducted in the field and the lab. Based on this classification, we found out that researchers have barely tested the measures in realistic scenarios. Some studies conducted controlled experiments, and others were less rigid experiments conducted in the laboratory or realistic environments. A large amount of controlled experiments is due to the academy's interest in knowing the real impacts in a controlled manner of the relationship between cognitive load and software artifacts.

Moreover, 6% (5/63) of primary studies were a proposal of solutions to cognitive load in software engineering. For instance, Peitek et al. [36] proposed a tool that integrates fMRI and eye-tracking to support the experiments in the field of software engineering, aiming to measure the cognitive load. Only 2% (1/63) proposed replicating a previous study, and 2% (1/63) validated the theories regarding cognitive load on software engineering. Of primary studies, only 3% (2/63) were experience papers in which authors described their past experiences with cognitive load. Only 3% (2/63) were philosophical papers that specify new visions and consequences.

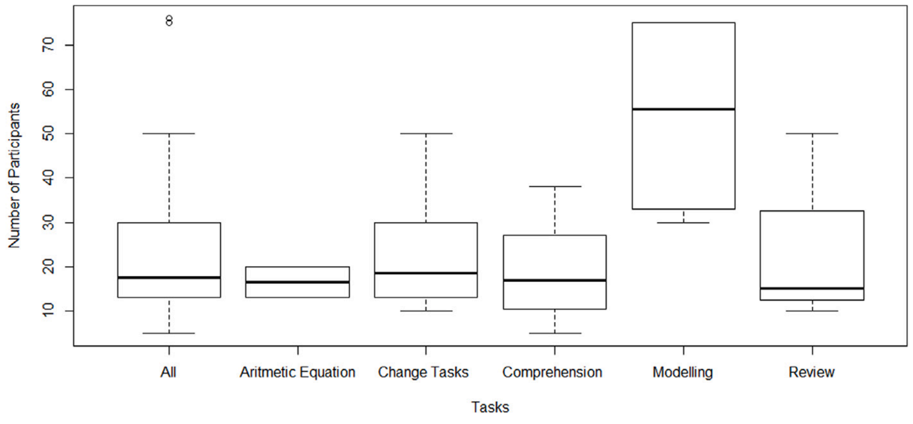


**Fig. 3.** The distribution of participants according to tasks.

**Table 22**
List of primary studies which were classified as validation studies.

| Validation studies | Primary studies |
|---|---|
| Controlled experiments | [S01] [S03] [S04] [S05] [S09] [S11] [S13] [S14] [S16] [S17] [S18] [S22] [S25] [S26] [S27] [S31] [S40] [S41] [S42] [S43] [S44] [S45] [S46] [S47] [S49] [S56] [S57] [S62] |
| Experimental lab studies | [S06] [S07] [S08] [S21] [S29] [S34] [S35] [S36] [S37] [S38] [S48] [S50] [S51] [S52] [S53] [S55] [S58] [S59] [S60] [S61] |
| Experimental studies in the field environment | [S19] [S24] [S39] [S32] |

Table 21 presents a tendency of primary studies to adopt validation studies methodology. The concentration of studies in this area demonstrates researchers are testing the viability of using the measures. This limitation is due to the difficulty of researchers in evaluating these measures outside of controlled environments. As it is possible to notice, in Table 22, only four validation studies performed evaluations in the field environment, i.e., they assessed the viability of measuring cognitive load related metrics in environments simulated a real work routine.

Based on the number of proposed solutions, there is a tendency that the integration between industry and academia remains limited. So far, researchers built tools for scientific analysis, but researchers did not provide the integration solutions that could be tested in practice for industry professionals to replicate. A limited number of studies replicated other researches. The lack of replication studies is a well-known problem in software engineering. Researchers have difficulties in accessing the datasets and replication packages. Psychophysiological data is sensitive and generally confidential. The lack of experience papers shows a limitation of software engineering researches in reporting errors and success stories from the past. The lack of these studies implies that new researchers are faded to repeat mistakes that experienced researchers already know how to solve.

Finally, researchers in software engineering also produced a limited number of philosophical articles. The lack of philosophical papers means that researchers did not structure the knowledge concerning cognitive load in software engineering the academia produced. For example, a framework would make it possible to define which measures and which processes future researchers could follow to organize how software engineering deals with a cognitive load.

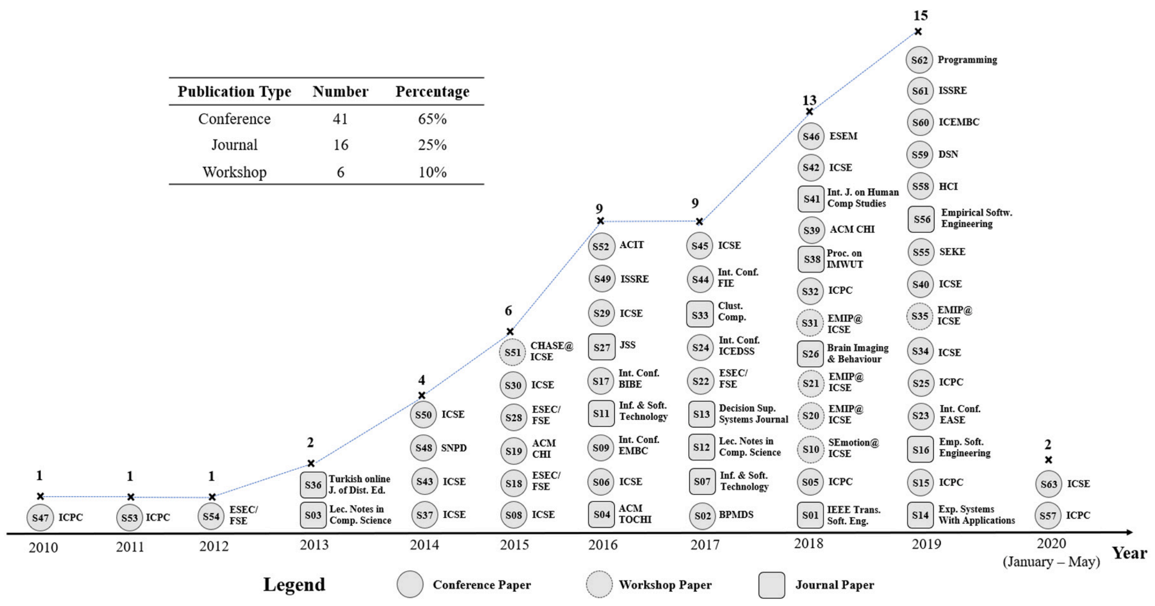
### *4.10. RQ10: Where have the studies been published?*

Fig. 4 shows a ten-year chronological plot of the primary studies. This work classifies these studies as conference papers, journal articles, or workshop paper.

**Number of publications**. Each published primary study adds one point to the total of the number of publications by year. Fig. 4 contains a dashed line that resumes the number of published primary studies per year. The largest number of publications occurred in 2018 and 2019. Thus, it is possible to notice the distribution and where the authors disseminated their research. Although the search strategy considered studies published until May 2020, the research focusing on cognitive load has significantly increased in software engineering research just after the release of technologies such as NeuroSky [52], and Emotiv [53]. Articles published after 2014 mentioned in these technologies frequently. Thus, the research on the measurement of the cognitive load of software developers has been steadily rising.

**Trends.** The number of studies has continuously grown since 2013 for each year, except 2017, which maintained the same amount of studies from the previous year. The most productive period was from 2017 until 2019, concentrating on 59% (37/63) of studies in conferences, journals, and workshops. During this period, five studies were published in workshops about the cognitive load in software engineering. The publication in workshops indicates a trend in discussing future challenges and emerging results with more frequency. The primary studies were also published in premier journals.[2] We found ten studies [S01] [S04] [S07] [S11] [S14] [S16] [S26] [S27] [S41] [S56] published on high impact journals (IEEE TSE, IST, Expert Systems with Applications, JSS, IJHCS, and Empirical Software and Engineering). The previous study [20] did not identified primary studies in journal focused in software engineering (IEEE Trans. on Software Engineering, Information and Software Technology, Journal of System and Software, and Empirical Software and Engineering). The research in this area has a strong trend to be published on Conferences related to software engineering, including ICSE [S06] [S08] [S29] [S30] [S34] [S37] [S40] [S42] [S43] [S45] [S50] [S63] and International Conference on Program Comprehension [S05] [S15] [S25] [S32] [S47] [S53] [S57]. Moreover, 2019 was the year that had the highest number of articles produced. Investigating the cognitive load in software engineering is an emerging research field with a strong potential to keep rising. Thus there are still significant challenges and problems researches can solve.

[2] Journals with h5-median higher than 40 according to the Google Scholar (https://scholar.google.com/).



| Publication Type | Number | Percentage |
|---|---|---|
| Conference | 41 | 65% |
| Journal | 16 | 25% |
| Workshop | 6 | 10% |

**Fig. 4.** The research venues in which authors published their primary studies over the last years.

The Section 5 discusses the primary studies' contributions and future challenges.

**Research venues.** Fig. 4 also shows that authors tend to publish primary studies on conferences rather than journals. This trend means that measuring cognitive load is a new and emerging research topic in software engineering. The publications tend to rise in journals as authors progressively structure the cognitive load in software engineering and develop new philosophical knowledge, such as taxonomies, and frameworks for evaluating cognitive load in software engineering.

## *4.11. Classification schema*

Fig. 5 presents a classification schema representing the answers used inside of each research question addressed in this work. Thus, this classification schema represents the knowledge obtained through the studies analyzed in this Systematic Mapping Study. This classification schema groups the answers to their research questions respectively. The black and blue colors highlight the root and subcategories of this schema, respectively. The black color highlights the domain of research. Squares highlighted with the darker blue represents the main research questions, in particular from RQ1 to RQ9. The classification scheme organizes these research questions side-by-side horizontally—the horizontal direction switches between the respective research questions in this research. The vertical direction of this schema contains the subcategories of each research question. The pale-blue color highlights these subcategories. These categories are the following:

- **Cognitive load (RQ1):** The theories in which primary studies used to define cognitive load;
- **Sensors (RQ2):** The devices in which primary studies adopted to capture cognitive load data;
- **Cognitive load measures (RQ3):** The measurements used to quantify the cognitive load;
- **Machine learning techniques (RQ4):** The machine learning classifiers used with cognitive load data;
- **Purposes (RQ5):** The purposes of software engineering studies on measuring the cognitive load;
- **Tasks (RQ6):** The tasks primary studies choose to evaluate cognitive load on experimental settings;
- **Artifacts (RQ7):** Which artifacts primary studies selected to evaluate cognitive load;
- **Participants (RQ8):** The number of participants in which primary studies recruited to attend to the tasks/experiments; and
- **Research methods (RQ9):** Research methods that primary studies adopt to investigate the cognitive load.

We used the scheme in Fig. 5 to track whether there is a concentration of studies in the horizontal direction. We found a concentration of six primary studies [S05] [S31] [S46] [S56] [S59] [S61] that were validation studies and used multimodal sensors and multimodal metrics. They also defined cognitive load as mental effort. These studies recruited between 15 and 30 participants in programming tasks with code artifacts. They used cognitive load with the purpose to investigate code comprehension [S05] [S46] [S56], task difficulty [S31], code quality concerns [S61], and classification of cognitive load [S59]. Another factor in common between these primary studies is that they did not use machine learning techniques with the psychophysiological data. These studies focused on correlating areas of the brain with software activities. It was also evident that primary studies used brain imaging resources such as fNIRS and fMRI with the Eye-Tracker to identify the variation in brain areas related to software development tasks [S05] [S31] [S46] [S56]. Another evident cluster of works is those that analyze brain area activation in software programming. These studies exclusively adopted fMRI [S01] [S22] [S26] [S43] [S49] to investigate code comprehension [S01] [S22] [S43] and quality concerns [S26] [S49].

Researchers and industry can benefit from various forms of this classification schema. These benefits are listed below:

**Main benefits for researchers:** Researchers perceive that measuring the cognitive load of developers as an opportunity to investigate the human factors in software development tasks. Thus, researchers have an interest in conducting a series of empirical studies to investigate the validity and effectiveness of using these measures in software development tasks. Based on this schema, researchers could classify future studies on cognitive load measurements in software engineering, and replicate the proposed research by pinpointing new gaps and further challenges. Researchers could also use this schema as a starting point for adding new terminologies related to the cognitive load measures in software engineering. Finally, researchers could use the terms adopted in this schema to conduct further empirical studies.

**Main benefits for software industry:** In industry, there is a strong interest in using cognitive load measures to improve developers' well-being, productivity, and the quality of software artifacts. In this way,

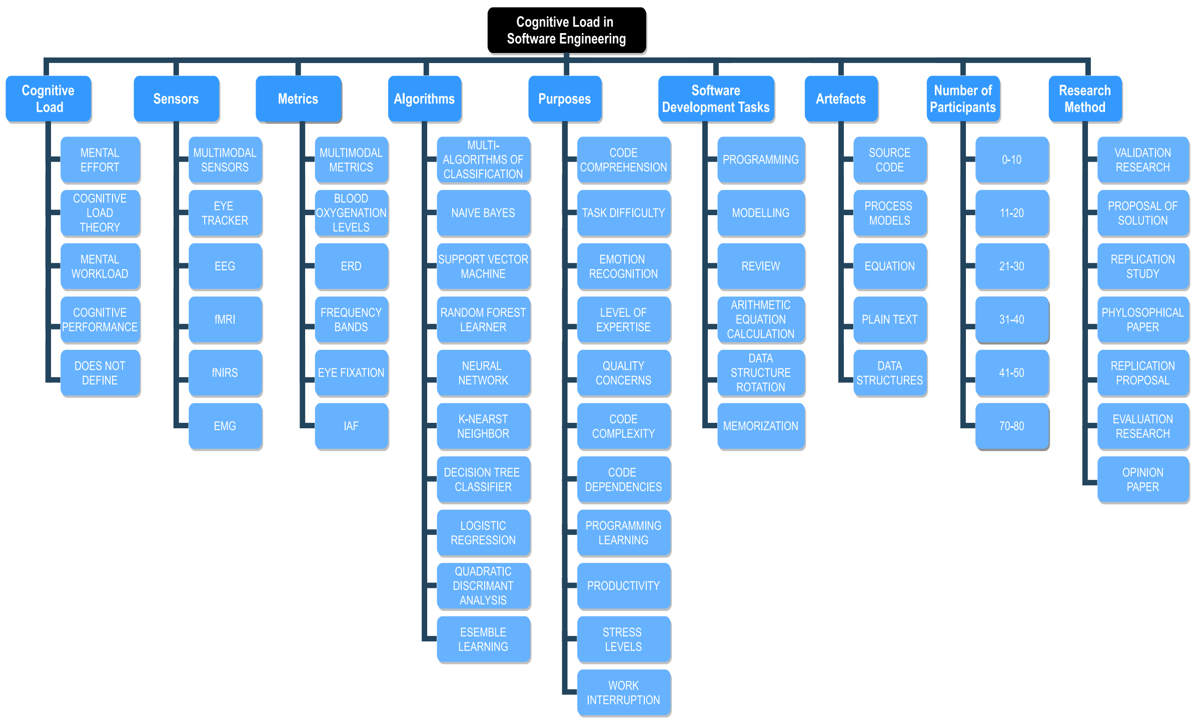


**Fig. 5.** Classification scheme of cognitive load in software engineering. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

developers and analysts would benefit from this schema by avoiding effort in surveying the technologies and methods related to the measures of developers' cognitive load. The software industry could build solutions to reduce developers' high cognitive load. Reducing the developers' cognitive load could avoid the insertion of bugs and improve their productivity.

## 5. Discussion

This section makes an additional discussion about the information contained in Fig. 6. Fig. 6 shows a bubble chart that organizes the primary studies through three dimensions. Each bubble has a triplet $(d_1, d_2, d_3)$ of associated data. The dimension $d_1$ represents the research method used and the contributions about the cognitive load on software engineering. The dimension $d_2$ presents the purposes in which primary studies measure cognitive load in software engineering. Moreover, $d_3$ consists of the number of studies, i.e., the size of the bubble. Section 5.1 presents a discussion about the bubble chart in Fig. 6. Section 5.2 presents the primary study contributions, highlighting its main findings and contradictions. Section 5.3 describes further challenges about the cognitive load on software engineering.

### 5.1. Bubble chart

The bubble chart in Fig. 6 shows that the primary studies distribution concentrates at the top of the upper quadrants of the research methods and the contributions facet. Specifically, the studies that measure the cognitive load in software engineering are validation studies that have contributed to empirical knowledge to understand the purposes of code comprehension and task difficulty.

Primary studies vaguely explore and evaluate relevant purposes presented in the figure. For example, for code complexity works only contributed with a method, which authors assessed through a validation study. To measure code complexity based on the developers' cognitive load is an ambitious goal. Fig. 6 shows that academia needs to produce knowledge both in practical and theoretical dimensions to measure developers' code complexity. This observation includes other purposes that can potentially impact the software industry, such as code quality concerns, programming learning, work interruption, and identification of code dependencies. The theoretical dimensions are about the theory that must emerge to support these purposes. For instance, new proposals of methods, creation of frameworks and taxonomies from philosophical papers, and empirical knowledge after validating the proposed methods. The practical dimension concerns integration of tools within the industry and the knowledge researchers can derive from it. For instance, researchers could extract new empirical knowledge from evaluation studies in industry and producing lessons learned from their experience.

Fig. 6 shows a strong tendency for primary studies conducting validation studies of cognitive load in the academic environment. Another factor that reinforces this bias in the academy is the absence of studies that evaluates tools in the industry. This bias happened even with studies aiming to solve program comprehension and task difficulty. For these purposes, researchers produced a reduced number of tools and evaluation studies in a realistic scenario, even after researchers produced a considerable amount of empirical knowledge and validation studies in academia.

### 5.2. Contributions

The selected primary studies focused on contributing mainly to five aspects of the cognitive load on software engineering. These contributions explicitly comprised of lessons learned, analysis, tools, empirical knowledge, and methods for measuring the cognitive load. We describe the primary studies contributions and their definition below:

**Lessons learned:** Some primary studies concerned with reporting lessons learned based on authors' previous experiences with experiments on cognitive load [S02] [S06] [S10] [S12] [S20] [S23] [S30]. Zimoch et al. [54] [S02] listed lessons learned about previous experiments about using eye-tracking on process model comprehension. The authors listed that the level of knowledge of participants is evident as the task level difficulty rises. Both studies [S02] [S23]

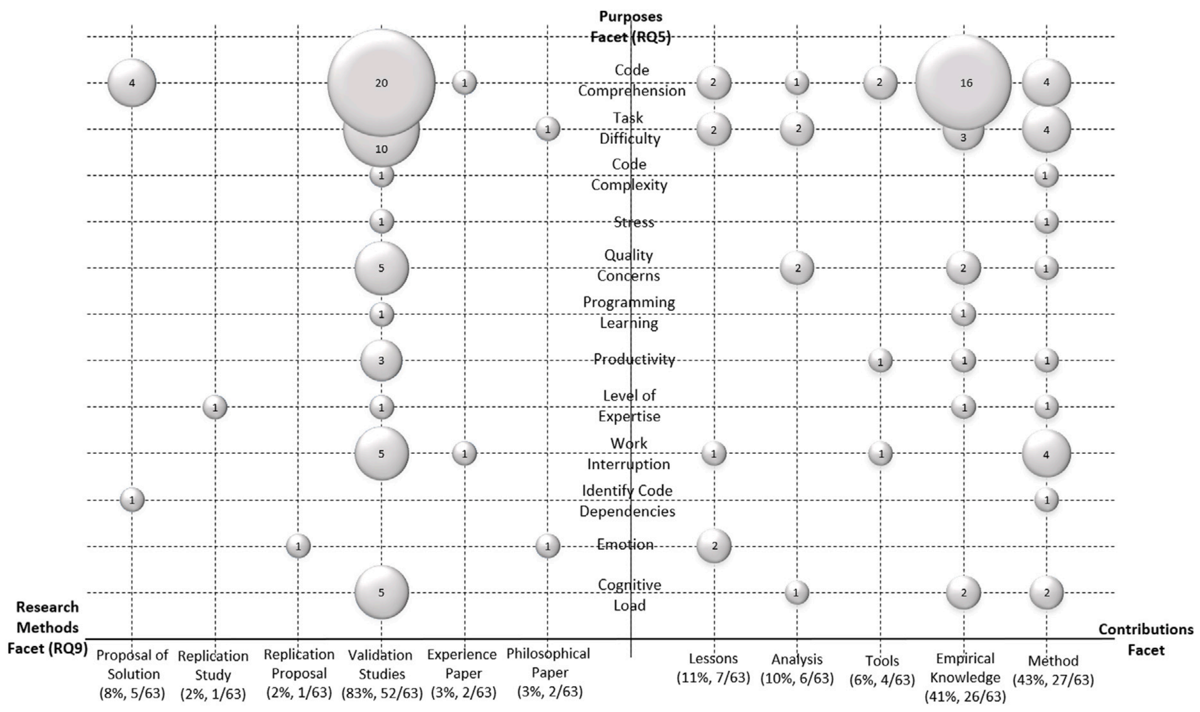


**Fig. 6.** Bubble chart that presents the relationship between three variables. The *axis-x* consists of the used research method (RQ9) and contributions. The *axis-y* represents the purpose (RQ5) of the primary studies. The bubbles' size represents the number of primary studies classified according to the criteria of the *axis-x* and *axis-y*.

agree and report that the context of tasks could influence the participants' perception and, then, negatively affect the results. For instance, a programming task formulated in c language for a java developer would only collect analysis focused on inexperienced developers in C-languages. Another important insight is that researchers attributed a better analysis of developers' program comprehension to data that was recovered by a combination of sensors [S06] [S10] [S12]. However, Molléri et al. [S23] suggests that most off-the-shelf sensors are not proper for experiments because sensors manufacturers of Neurosky and other commercial wrist bands projected this hardware aiming personal use. The author reported he obtained failed recordings from the EEG headset that had generated readings with missing information.

**Analysis:** This part presents the primary studies that conducted a comparative analysis of the effectiveness of cognitive load measures [S03] [S07] [S29] [S31] [S35] [S62]. Nourbakhsh et al. [S03] analyzed the impact of eye blink and galvanic skin response on the cognitive load classification. The authors concluded that combining GSR and eye blinks is more precise than using solely one of them to classify cognitive load levels. Borys et al. [S07] concluded that pupil dilation and EEG were not relevant for measuring the cognitive load. The correlation between the pupil dilation and the level of the developers' cognitive load was lower. This low correlation was due to the brightness level of the computer screen and the environment. It decreased the developers' pupils implying pupil variations in the different stimuli of the task. However, Peitek et al. [S31] evidenced that pupil dilation varied even with screen brightness. Duraisingam et al. [55] also disagreed with Borys et al. [S07], showing the effectiveness of EEG in measuring cognitive load. The results of Ishida et al. [S35] and Uwano et al. [S62] also diverged about Borys. Ishida et al. [S35] and Uwano et al. [S62] had evidenced the effectiveness of EEG in measuring cognitive load. The improvement must have happened because, in their study, the authors removed from the EEG noise and movement components, i.e., involuntary movements of eyes not related to the cognitive load analysis.

**Tools:** Some articles contributed with software tools to support the measurement of cognitive load in software engineering for research and practical applications [S15] [S38] [S51] [S63]. Some primary studies focused on proposing tools for experiments involving developers' cognitive load in the software engineering research field, such as CodersMUSE [S15] and VITALISE [S63]. The primary studies also provided solutions that the software industry can apply in practice. Flowlight [S32] and Collins [S38] aim to manage the developers' work interruption based on cognitive load indicators. These systems point when the developer can be interrupted or not. Emendo [S51] is a model that aims to monitor developers' cognitive load levels in real-time for managing and monitor developers' productivity.

**Empirical knowledge:** Some primary studies contributed with empirical knowledge. These studies focused on binding the cognitive load measures to the context of software engineering effectively [S01] [S04] [S05] [S09] [S11] [S13] [S16] [S17] [S22] [S26] [S36] [S40] [S41] [S43] [S44] [S45] [S46] [S47] [S48] [S49] [S50] [S52] [S53] [S56] [S57] [S59]. For instance, some primary studies provide strong evidence that specific brain areas were activated while developers comprehend source code. Peitek et al. [10] [S01], and Siegmund et al. [26] [S22] found that brain areas related to language processing (BA 21, BA 44, BA 47), attention (BA 6), and working memory (BA 6, BA 40) activated during code comprehension tasks. Castelhano et al. [37] investigated brain areas related to software bug detection. The same areas activated in relation to the Peitek et al. [10] [S01] and Siegmund et al. [26] [S22] was found. Moreover, Castelhano et al. [37] also found brain areas related to math processing (BA 9) were also activated. These areas were not found in Peitek et al. [10] [S01], and Siegmund et al. [26] [S22] because the task design does not require developers to process math equations. Overall, Siegmund et al. [26] [S22] also provides a state-of-art framework to investigate program comprehension with fMRI sensors. Furthermore, Krueger et al. [S57] and Floyd, Santander, and Weimer [S45] found the brain areas which activate while developers read a source code. Both researchers found that the brain activation pattern between interpreting a source code is different from the brain areas activated in the brain when developers read texts (reading tasks). These studies also serve as a guide for analyzing the impact of brain activation on different artifacts.

**Method:** It consists of articles that mainly contributed with methods or approaches to measure the cognitive load on software engineering [S08] [S14] [S18] [S19] [S21] [S24] [S25] [S27] [S28] [S32]

[S33] [S34] [S37] [S39] [S42] [S54] [S55] [S58] [S60] [S61]. These studies mainly systematized ways to relating the software artifacts and components to developers' cognitive load measures. The literature contributed with several methods to measure cognitive load in software engineering. Overall, it consists of machine learning approaches that researchers applied for software engineering. Müller & T. Fritz 2015 [S08] were capable of predicting developers' emotions (positive and negative emotions) with 71.36% of precision. There are contributions related to methods to identify specific quality concerns in source code. Müller & Fritz 2016 [S24] [S61] trained a Random Forest machine learning technique using measures such as skin temperature, heart rate variability, and respiratory rate measures features for classifying types of quality concerns. This technique was able to identify the types of quality concerns, such as coding style, bugs, missing tests, and inadequate comments. The precision of classifying when developers are prone to insert parts of code that contain bugs reached a precision of 50%. Couceiro et al. [S61] used an Eye-Tracker combined with heart rate variability to develop a method to annotate the cognitive load information about a specific token' code. This method indicates that technologies can already correlate software defects and pinpoint code part where developers are spending more cognitive load.

### 5.3. Challenges for future research

This section presents some challenges derived from the selected Primary Studies and the analyzed contributions found in the last section. This work describes the future challenges as follows:

**(1) Report lessons learned about the experience of measuring the cognitive load.** Reporting lessons learned is an essential practice because they report solutions from good and bad past experiences. Researchers and practitioners are interested in reports of lessons learned because it provides useful and wrong methods in advance. According to Fig. 6, there is a lack of lessons learned in practically all the applications identified in this work (RQ5). Reporting lessons learning of past experiences is not a common practice in the software engineering research field.

The lack of lessons learned is also a consequence of a lack of replication studies. In these studies, the authors generally choose to repeat a previous study applying new methods. However, there is not a well-recognized guideline for reporting replication studies in software engineering [56]. Therefore, there are no well-defined guidelines to conduct this kind of research. This lack implies that researchers do not know what academia considers as a concrete contribution from replication studies. In replication studies, authors could report some deficiencies or improvements about sensors, metrics, or methods when matching the current results with the earlier version of the research. Supplying this lack of lessons learned is a significant challenge to address. Another possible solution is to extract and infer lessons learned from validation studies academia already produced applying Ground Theory. The annotations could be sent through surveys for the involved authors to confirm, add, or revise.

**(2) Analysis on the effectiveness of psychophysiological metrics in pointing cognitive load level.** To analyze the performance on metrics related to cognitive load is required to determine which of these metrics has better efficiency. For instance, Section 5.2 reports that researchers contributed with an analysis aiming to point which psychophysiological indicators would be most relevant in pointing out cognitive load. However, these analyses were limited to the findings pointed out in Section 5.2, specifically for the effectiveness of GSR, Eye trackers, and EEG-based measures. It is a range of metrics that are not even close to those listed in RQ3.

Therefore, researchers did not explore the effectiveness of metrics related to heart rate, respiration rate, and skin temperature in the body of knowledge already produced. In particular, researchers have the challenge of filling the gap of the lacking analysis regarding all purposes that lacks circles in Fig. 6. Fig. 6 shows the lack of study analysis for most purposes. Moreover, researchers explored the effectiveness of the electroencephalogram a little in existing analyses. Further analysis of the effectivity of the electroencephalogram also must be done. Researchers can analyze the usability and acceptability of such technologies by the developers in realistic scenarios.

**(3) Develop a tool for supporting the cognitive load monitoring in realistic-settings.** Tools are one of the most visible and expected results by the software industry. This research field promises using the cognitive load level as a predictor of code quality problems, improving the conditions for maintaining software. However, researches produced a few tools for this purpose. Instead, most of the primary studies aimed to support the analysis of multimodal cognitive load measurements on lab studies. Another example is the work interruption management system, i.e., FlowLight. There is a challenge to produce tools to indicate the high cognitive load of developers. Tools in software engineering could reduce developers' cognitive load by recommending less mentally demanding tasks to them. Consequently, measuring developers' cognitive load could reduce the quality concerns in source code [23], improve code comprehension [57], and reduce developers' task difficulty [8].

Therefore, researchers could use the body of knowledge provided in primary studies for developing tools for applying in realistic scenarios of the software industry. The first-hand solution for this would be the proposal of an integrated development environment (IDE) tool. Researchers could customize the IDE Eclipse with a plugin for collecting the sensors' information and embed the already existent classifiers for quality concern, task difficulty, and expertise levels on this IDE to support developers.

**(4) Empirical knowledge about cognitive processes in software engineering tasks.** Conducting empirical studies in software engineering is a challenging task [57]. Most of the contributions of the software engineering research field are empirical knowledge. Most of them concerned with code comprehension, specifically. Still, there is a lack of empirical knowledge about developers' cognitive processes on task difficulty, code complexity, stress levels, work interruption, identification of code dependencies, and the effects of emotions on the software development environment. Therefore, future empirical knowledge could focus on these software engineering purposes.

For this, the study of Siegmund et al. [26] and Crk and Kluthe [3] are good examples and can be used as a reference to structure future controlled experiments. Thus problems that lack investigations such as how the cognitive load affects or is affected according to the task difficulty level, code complexity, and others could be addressed. Furthermore, researchers could investigate more about the developers' brain functions. These include examining the brain areas activated during an increasing level of code complexity. Researchers could then explore the relations of code complexity on cognitive load indicators. As the lack of replication studies are evident, future research could replicate the already produced empirical studies about code comprehension and quality concerns.

**(5) A method to measure cognitive load in software engineering.** Academia applied many methods to measure the cognitive load of software developers. Researchers usually measured the developers' cognitive load through the range of metrics obtained from the vast combination of sensors. This range of metrics used demonstrates that measuring the developers' cognitive load has validity among several metrics. Among the measures used there is Event-Related Desynchronization (ERD). Crk and Kluthe [3] calculated the ERD based on the developers' brainwave. Crk and Kluthe [3] obtained the ERD calculating the mean of the cognitive load applied during their task activity over developers resting periods. Thus, there is a consolidated way to materialize methods for measuring the developer's cognitive load. The existence of cognitive load metrics enables researchers to build and propose a specific measure of developers' cognitive load.

Therefore, researchers should focus on proposing methods for measuring the developers' cognitive load for the listed purposes presented in Fig. 6. Fig. 6 does not have a proposed method for measuring cognitive load for programming learning purposes. This lack is critical because sophomore students usually have difficulty in learning new programming skills. To fill this gap, researchers should develop tools that assist the progressive gained knowledge from students. Based on this, this tool should recommend appropriate tasks to improve the students' skills. Only one primary study suggested a method to identify the code complexity level based on cognitive load measures. Researchers could fine-tune the precision of techniques that use developers' cognitive load for measuring the code complexity by testing more cognitive load indicators, e.g., those presented in Section 4.1. Researchers proposed only one method to classify code quality concerns and productivity. For this, researchers should apply classification techniques (Section 4.3) on a more significant and diverse dataset, i.e., containing different software engineering tasks of distinct programming languages and a wide range of software developers with various levels of experience.

## 6. Threats to validity

This section presents the measures taken for mitigating the threats to validity. In particular, we mitigated these threats concerning the construct, internal, statistical validity, and external threats [38,58].

**Construct validity.** This threat concerns the measures that assure we followed proper operational instructions that support the statements in which research questions investigate. It consists of mitigating the risks that could prevent our SMS investigates what it claims to be investigating. The inclusion of improper terms in the search string we used in the automatic search may cause the exclusion of some studies. We followed well-known methods [16,18] for extracting the appropriate synonyms and the main keywords for the search string. This measure also attenuates a gap between the keywords of potential studies keywords and our search string. We also considered an extensive list of search engines related to the field of research, including Google Scholar, and Scopus to mitigate the risk of venue inaccessibility. Furthermore, non-peer-reviewed materials (gray literature), such as patent specifications, technical reports, which are absent from review processes with scientific nature, were excluded from the selection process. We followed a rigorous search protocol for systematic mapping studies to maximize the inclusion of relevant papers and avoid including wrong studies [16,18]. These guidelines define the creation of inclusion and exclusion criteria and a multi-step filtering process for selecting primary studies. We added to this process the snowballing methodology to increase the coverage of our search. The snowballing is a process for searching for more studies in the references from a list of potential primary studies. According to Petersen et al. [17] and Kitchenham and Charters [18] the combination of manual with automatic searches based on keywords helps cover possible gaps present in the search string terms. The usage of this rigorous and systematic methodology also mitigates any threat concerning the possibility of the definition of improper inclusion and exclusion criteria and the adoption of an incorrect search method [58].

**Internal validity.** This category of threat mitigates whether the data used to ascertain conclusions are internally valid [50]. Thus, the measures to reduce risks for avoiding deriving results from inadequate data. Only one author conducted the selection and filtering process. This conduction can threaten the validity of studies' inclusion due to the authors' subjectivity in the primary studies selection. To mitigate this problem, the author strictly followed the well-established guidelines for filtering studies [16,18,38]. In particular, during the studies selection, the author responsible for the filtering process also read the introduction of articles in the case neither the title nor the abstract present enough details to consider the study for inclusion in the next step. Another threat to internal validity is the presence of duplicate studies. The author in charge of the filtering process identified duplicated studies using the Mendeley tool. This tool matches studies in duplicate by title. In the case of matching results be positive, the author removed the duplicated ones after checking the abstracts of both studies.

**Conclusion validity.** This category of threat discusses measures taken to avoid biased conclusions of the obtained results. We made all the results and analysis based on the literature about systematic mapping methodology [16,18]. Two authors in parallel conducted the data extraction from the selected studies. We applied coding to define the study categories, a method we borrowed from the grounded theory. The authors conducted a pilot data extraction from five primary studies, and then they checked the results of one another and resolved their disagreements. Correctly classifying and analyzing the selected studies was a constant concern during this research. For this, we dedicated a significant effort to examine the primary studies. After the analysis, we focused on avoiding the fishing problem. Consequently, we did not assert conclusions before analyzing the results. For this, we formulated the study conclusions only after examining the results we collected [38].

**External validity.** This category of threat deals with risks concerning the generalization of the findings of this study [58]. For instance, the inaccessibility of articles could jeopardize the representability of our sample. The university guaranteed access to the articles used in this research, turning the paper inaccessibility a reduced risk. Another threat to external validity is the range of years that the inclusion criteria considered to select potential study candidates. The upper and lower bounds could lead to excluding relevant studies. We did not define any time constraint to retrieve potentially relevant study candidates in search engines (Table 4).

## 7. Conclusion

We conducted a Systematic Mapping Study about measures of cognitive load in software engineering. We selected 63 primary studies from initially 4175 potential candidates. We followed a rigorous selection process with eight steps. In this process, we also conducted a snowballing process to check the references and guarantee the inclusion of relevant primary studies. Moreover, we classified the selected studies according to ten dimensions, which include the definition of cognitive load, sensors, metrics, machine learning algorithms, purposes, tasks, artifacts, participants, research methods, and research venues. The concluding remarks of each dimension are the following:

1. **Cognitive load:** Primary studies tends to not define the term cognitive load;
2. **Sensors:** Primary studies that relied on one sensor adopted the Eye-Tracker or the electroencephalogram;
3. **Metrics:** The works applied various metrics to measure cognitive load to turn the cognitive load measure more accurate;
4. **Machine learning algorithms:** The primary studies tend to adopt more than one classification algorithm with Support Vector Machines (SVM), and Naïve Bayes (NB) being more frequent;
5. **Purposes:** Researchers were concerned with how developers understand the source code (code comprehension) by analyzing which brain regions were involved in this cognitive process;
6. **Tasks:** Majority of primary studies focused on measuring the cognitive load during programming tasks;
7. **Artifacts:** Primary studies adopted traditional and well-known languages such as Java, and C#, i.e., programming languages in which participants already were skilled;
8. **Number of participants:** Recruiting participants is difficult because the nature of experiment differs and requires specific participants' profile;
9. **Research methods:** Primary studies sought to investigate the effects and viability of cognitive load measures in software engineering activities rather than proposing solutions;
10. **Research venues:** We frequently found most of primary studies published in conferences such as ICSE, and ICPC.

Moreover, we generated a classification scheme of cognitive load in software engineering using each dimension's answers. Next, this research discussed the primary studies' contributions and outlined future challenges identified in primary studies. From this, we can outline some future directions: (1) A definitive method for measuring the cognitive load of practitioners; (2) Future studies must advance the investigations about the prediction of unproductive periods of developers, e.g., identifying when their cognitive load levels are prone to produce bugs or errors; and (3) Develop tools to enable the usage and evaluation of cognitive load in realistic-scenarios. Finally, this work is an initial step in the research of cognitive load measures in software engineering.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. We thank the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) grant number 313285/2018-7 for partially funding this research.

## Appendix. Selected primary studies

**S01** N. Peitek, J. Siegmund, S. Apel, C. Kästner, C. Parnin, A. Bethmann, T. Leich, G. Saake, & A. Brechmann "A Look into Programmers' Heads." IEEE Transactions on Software Engineering, 2018.

**S02** M. Zimoch, R. Pryss, J. Schobel, & M. Reichert. "Eye Tracking Experiments on Process Model Comprehension: Lessons Learned." In Ent., Business-Process and Inf. Systems Modeling. BPMDS 2017. Lec. Notes in Business Inf. Proc., v 287.

**S03** N. Nourbakhsh, Y. Wang, & F. Chen. "GSR and blink features for cognitive load classification," In IFIP Conf. on HCI, 2013, pp. 159–166.

**S04** I. Crk, T. Kluthe, & A. Stefik. "Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes". ACM TOCHI. 23, 1, 2015.

**S05** S. Fakhoury, Y. Ma, V. Arnaoudova, & O. Adesope "The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load." ICPC, 2018.

**S06** T. Fritz & S. C. Müller, "Leveraging Biometric Data to Boost Software Developer Productivity," SANER, 2016, pp. 66–77.

**S07** M. Borys, M. Plechawska-Wójcik, M. Wawrzyk & K. Wesołowska. "Classifying Cognitive Workload Using Eye Activity and EEG Features in Arithmetic Tasks." Information and Software Technology, 2017, pp. 90–105.

**S08** S. C. Müller & T. Fritz, "Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress," ICSE, 2015, pp. 688–699.

**S09** I. Crk & T. Kluthe, "Assessing the contribution of the individual alpha frequency (IAF) in an EEG-based study of program comprehension," EMBC, 2016, pp. 4601–4604.

**S10** D. Girardi, F. Lanubile, N. Novielli, & D. Fucci. 2018. "Sensing developers' emotions: the design of a replicated experiment," SEmotion, 2018, pp. 51–54.

**S11** R. Petrusel, J. Mendling, & H. A. Reijers. "Task-specific visual cues for improving process model understanding." Information and Software Technology, v. 79, 2016, pp. 63–78.

**S12** R. K. Minas, R. Kazman, & E. Tempero. "Neurophysiological Impact of Software design processes on software developers." In Int. Conf. on Augmented Cognition, 2017, pp. 56–64.

**S13** R. Petrusel, J. Mendling & H. A. Reijers. "How visual cognition influences process model comprehension." Decision Support Systems, V. 96, 2017, pp. 1–16.

**S14** M. Tallon, M. Winter, R. Pryss, K. Rakoczy, M. Reichert, M. W. Greenlee, & U. Frick. "Comprehension of business process models: Insight into cognitive strategies via eye tracking." Expert Systems with Applications, V. 136, 2019, pp. 145–158.

**S15** N. Peitek, S. Apel, A. Brechmann, C. Parnin, & J. Siegmund. "CodersMUSE: multi-modal data exploration of program-comprehension experiments." ICPC, 2019, pp. 126–129.

**S16** T. Baum, K. Schneider, & A. Bacchelli. "Associating working memory capacity and code change ordering with code review performance." Empirical Software Engineering, 2019, v. 24, n. 4, pp. 1762–1798.

**S17** S. Lee, A., Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, & H. Lim. "Comparing programming language comprehension between novice and expert programmers using EEG analysis." BIBE, 2016, pp. 350–355.

**S18** K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, & T. Fritz. "Tracing software developers' eyes and interactions for change tasks." ESEC/FSE, 2015, pp. 202–213.

**S19** M. Züger & T. Fritz. "Interruptibility of Software Developers and its Prediction Using Psycho-Physiological Sensors." CHI. 2015, pp. 2981–2990.

**S20** N. Peitek, J. Siegmund, C. Parnin, S. Apel, & A. Brechmann. "Toward conjoint analysis of simultaneous eye-tracking and fMRI data for program-comprehension studies." EMIP, 2018.

**S21** M. Behroozi & C. Parnin. "Can we predict stressful technical interview settings through eye-tracking?." EMIP, 2018.

**S22** J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, & A. Brechmann, "Measuring neural efficiency of program comprehension." In ESEC/FSE, 2017, pp. 140–150.

**S23** J. S. Molléri, I. Nurdiani, F. Fotrousi, & K. Petersen. "Experiences of studying Attention through EEG in the Context of Review Tasks." EASE, 2019, pp. 313–318.

**S24** A. Duraisingam, R. Palaniappan, & S. Andrews "Cognitive task difficulty analysis using EEG and data mining." ICEDSS, 2017, pp. 52–57.

**S25** D. Fucci, D. Girardi, N. Novielli, L. Quaranta, and F. Lanubile. "A replication study on code comprehension and expertise using lightweight biometric sensors." In ICPC, 2019.

**S26** J. Castelhano, I. C. Duarte, C. Ferreira, J. Duraes, H. Madeira, & M. Castelo-Branco. "The role of the insula in intuitive expert bug detection in computer code: an fMRI study." Brain Imaging and Behavior, V. 13, I. 3, pp. 623–637.

**S27** K. Kevic, B.M. Walters, T.R. Shaffer, B. Sharif, D.C. Shepherd, & T. Fritz "Eye gaze and interaction contexts for change tasks – Observations and potential." JSS, V. 128, 2017, pp. 252–266.

**S28** M. Konopka. "Combining eye tracking with navigation paths for identification of cross-language code dependencies." ESEC/FSE, 2015, pp. 1057–1059

**S29** S. C. Müller & T. Fritz. “Using (bio) metrics to predict code quality online.” ICSE. 2016, pp. 452–463.

**S30** S. C. Müller “Measuring software developers’ perceived difficulty with biometric sensors.” ICPC, 2015, pp. 887–890.

**S31** N. Peitek, J. Siegmund, C. Parnin, S. Apel, & A. Brechmann “Beyond gaze: preliminary analysis of pupil dilation and blink rates in an fMRI study of program comprehension.” In EMIP, 2018.

**S32** M. Züger & T. Fritz “Sensing and supporting software developers’ focus.” In ICPC, 2018.

**S33** S. Lee, D. Hooshyar, H. Ji, K. Nam, & H. Lim. “Mining biometric data to predict programmer expertise and task difficulty.”, Cluster Computing, 2017, pp. 1–11.

**S34** R. Couceiro, G. Duarte, J. Durães, J. Castelhano, C. Duarte, C. Teixeira, M. C. Branco, P. Carvalho, & H. Madeira “Biofeedback augmented software engineering: monitoring of programmers’ mental effort.” ICSE: NIER, 2019, pp. 37–40.

**S35** T. Ishida, & H. Uwano “Synchronized analysis of eye movement and EEG during program comprehension.” EMIP, 2019, pp. 26–32.

**S36** M. P. Uysal. “Towards the use of a novel method: The first experiences on measuring the cognitive load of learned programming skills.” Turkish Online Journal of Distance Education, v.14, n. 1, 2013, pp. 166–184.

**S37** T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, & M. Züger “Using psycho-physiological measures to assess task difficulty in software development.” ICSE, 2014, pp. 402–413.

**S38** F. Schaule, J. O. Johanssen, B. Bruegge, & V. Loftness “Employing Consumer Wearables to Detect Office Workers’ Cognitive Load for Interruption Management.” Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Tech., v. 2, n. 1, 2018, pp. 32:1–32:20.

**S39** M. Züger, S. C. Müller, A. N. Meyer, & T. Fritz “Sensing interruptibility in the office: A field study on the use of biometric and computer interaction sensors.” CHI, 2018, pp. 591.

**S40** Y. Huang, X. Liu, R. Krueger, T. Santander, X. Hu, K. Leach, & W. Weimer “Distilling neural representations of data structure manipulation using fMRI and fNIRS.” ICSE, 2019, pp. 396–407.

**S41** M. V. Kosti, K. Georgiadis, D. A. Adamos, N. Laskaris, D. Spinellis, & L. Angelis “Towards an affordable brain computer interface for the assessment of programmers’ mental workload.” Int. Journal of HCI, 2018, v. 115, pp. 52–66.

**S42** M. Behroozi, A. Lui, I. Moore, D. Ford, & C. Parnin. “Dazed: measuring the cognitive load of solving technical interview problems at the whiteboard.” ICSE: NIER, 2018, pp. 93–96.

**S43** J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, & A. Brechmann “Understanding understanding source code with functional magnetic resonance imaging.” In ICSE, 2014, pp. 378–389.

**S44** M. K. C. Yeh, D. Gopstein, Y. Yan, & Y. Zhuang “Detecting and comparing brain activity in short program comprehension using EEG.” IEEE Front. in Education Conf., 2017, pp. 1–50.

**S45** B. Floyd, T. Santander, & W. Weimer “Decoding the representation of code in the brain: an fMRI study of code review and expertise.” ICSE, 2017, pp. 175–186.

**S46** N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. C. Hofmeister, A. Brechmann “Simultaneous measurement of program comprehension with fMRI and eye tracking: a case study.” ESEM, 2018.

**S47** B. Sharif, & J. I. Maletic, “An eye tracking study on camelcase and under_score identifier styles.” ICPC, 2010, pp. 196–205.

**S48** Y. Ikutani & H. Uwano “Brain activity measurement during program comprehension with NIRS” SNPD, 2014, pp. 1–6.

**S49** J. Duraes, H. Madeira, J. Castelhano, C. Duarte, & M. C. Branco “WAP: Understanding the Brain at Software Debugging” ISSRE, 2016, pp. 87–92.

**S50** T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, D. M. & German “Quantifying programmers’ mental workload during program comprehension based on cerebral blood flow measurement: a controlled experiment.” ICSE, 2014, pp. 448-451.

**S51** S. Radevski, H. Hata & K. Matsumoto “Real-time monitoring of neural state in assessing and improving software developers’ productivity” CHASE, 2015, pp. 93–96.

**S52** A. Yamamoto, H. Uwano, & Y. Ikutani. “Programmer’s electroencephalogram who found implementation strategy.” ACIT-CSII-BCD, 2016, pp. 164-168.

**S53** C. Parnin “Subvocalization-toward hearing the inner thoughts of developers” ICPC, 2011, pp. 197–200.

**S54** J. Siegmund, A. Brechmann, S. Apel, C. Kästner, J. Liebig, T. Leich, & G. Saake. “Toward measuring program comprehension with functional magnetic resonance imaging.” ESEC/FSE, 2012, pp. 1-4.

**S55** R. Ikramov, V. Ivanov, S. Masyagin, R. Shakirov, I. Sirazidtinov, G. Succi, & O. Zufarova. “Initial evaluation of the brain activity under different software development situations.” SEKE, 2018.

**S56** S. Fakhoury, D. Roy, Y. Ma, V. Arnaoudova, & O. Adesope. “Measuring the impact of lexical and structural inconsistencies on developers’ cognitive load during bug localization” Emp. Soft. Engineering, 1-39, 2019.

**S57** R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer, & K. Leach “Neurological Divide: An fMRI Study of Prose and Code Writing”, ICSE, 2020.

**S58** R. Palaniappan, A. Duraisingam, N. Chinnaiah, & M. Murugappan “Predicting Java Computer Programming Task Difficulty Levels Using EEG for Educational Environments.” HCI, 2019, pp. 446–460.

**S59** R. Couceiro, G. Duarte, J. Durães, J. Castelhano, C. Duarte, C. Teixeira, M. C. Branco, P. Carvalho & H. Madeira “Pupillography as Indicator of Programmers’ Mental Effort and Cognitive Overload” DSN, 2019, pp. 638–644.

**S60** J. Medeiros, R. Couceiro, J. Castelhano, M. Castelo Branco, G. Duarte, C. Duarte, J. Durães, H. Madeira, P. Carvalho, C. Teixeira, “Software code complexity assessment using EEG features.” International Conference of the IEEE EMBC, 2019, pp. 1413–1416.

**S61** R. Couceiro, R. Barbosa, J. Duráes, G. Duarte, J. Castelhano, C. Duarte, C. Teixeira, N. Laranjeiro, J. Medeiros, P. Carvalho, H. Madeira. “Spotting Problematic Code Lines using Nonintrusive Programmers’ Biofeedback” ISSRE, pp. 93–103, 2019.

**S62** T. Ishida & H. Uwano “Time series analysis of programmer’s EEG for debug state classification” Programming, 2019, pp. 1–7.

**S63** D. Roy, S. Fakhoury & V. Arnaoudova “VITALSE: Visualizing Eye Tracking and Biometric Data.” ICSE, 2020.

**Extensions of Primary Studies:** RQ1 of [S30] extends [S37], [S08] extends RQ2 of [S30], [S29] extends RQ3 of [S30], [S06] extends challenges and future works of [S30], [S01] extends [S22], [S22] replicates [S43], [S39] extends [S19], [S56] extends [S05], [S27] extends [S18].

## References

[1] J. Sweller, Cognitive load theory, in: Psychology of Learning and Motivation, Vol. 55, Elsevier, 2011, pp. 37–76.
[2] W. Klimesch, EEG alpha and theta oscillations reflect cognitive and memory performance: a review and analysis, Brain Research Reviews 29 (2–3) (1999) 169–195.
[3] I. Crk, T. Kluthe, Assessing the contribution of the individual alpha frequency (IAF) in an EEG-based study of program comprehension, in: 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2016, pp. 4601–4604.
[4] I. Crk, T. Kluthe, A. Stefik, Understanding programming expertise: An empirical study of phasic brain wave changes, ACM Transactions on Computer-Human Interaction (TOCHI) (ISSN: 1073-0516) 23 (1) (2015) 2:1–2:29.
[5] A. Sinharay, D. Chatterjee, A. Sinha, Evaluation of different onscreen keyboard layouts using EEG signals, in: 2013 IEEE International Conference on Systems, Man, and Cybernetics, 2013, pp. 480–486.
[6] R.K. Minas, R. Kazman, E. Tempero, Neurophysiological impact of software design processes on software developers, in: Augmented Cognition. Enhancing Cognition and Behavior in Complex Human Environments, Springer International Publishing, 2017, pp. 56–64.
[7] S. Lee, D. Hooshyar, H. Ji, K. Nam, H. Lim, Mining biometric data to predict programmer expertise and task difficulty, Cluster Computing (2017) http://dx.doi.org/10.1007/s10586-017-0746-2.
[8] T. Fritz, A. Begel, S.C. Müller, S. Yigit-Elliott, M. Züger, Using psycho-physiological measures to assess task difficulty in software development, in: Proceedings of the 36th international conference on software engineering, in: ICSE 2014, 2014, pp. 402–413.
[9] T. Fritz, S.C. Müller, Leveraging biometric data to boost software developer productivity, in: 2016 IEEE 23rd international conference on software analysis, evolution, and reengineering (SANER), Vol. 5, 2016, pp. 66–77.
[10] N. Peitek, J. Siegmund, C. Parnin, S. Apel, A. Brechmann, Toward conjoint analysis of simultaneous eye-tracking and fMRI data for program-comprehension studies, in: Proceedings of the Workshop on Eye Movements in Programming, EMIP '18, ACM, New York, NY, USA, 2018, pp. 1:1–1:5.
[11] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk, R. Oliveto, Automatically assessing code understandability, IEEE Transactions on Software Engineering 47 (3) (2021) 595–613.
[12] M.V. Kosti, K. Georgiadis, D.A. Adamos, N. Laskaris, D. Spinellis, L. Angelis, Towards an affordable brain computer interface for the assessment of programmers' mental workload, International Journal of Human-Computer Studies 115 (2018) 52–66.
[13] Q. Gui, M.V. Ruiz-Blondet, S. Laszlo, Z. Jin, A survey on brain biometrics, ACM Computing Surveys (CSUR) 51 (6) (2019) 112:1–112:38, http://dx.doi.org/10.1145/3230632, URL: http://doi.acm.org/10.1145/3230632.
[14] A. Bablani, D.R. Edla, D. Tripathi, R. Cheruku, Survey on brain-computer interface: An emerging computational intelligence paradigm, ACM Computing Surveys (CSUR) 52 (1) (2019) 20:1–20:32.
[15] S. Fakhoury, Y. Ma, V. Arnaoudova, O. Adesope, The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load, in: 2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC), 2018.
[16] B.A. Kitchenham, D. Budgen, O. Pearl Brereton, Using mapping studies as the basis for further research - a participant-observer case study, Information and Software Technology 53 (6) (2011) 638–651.
[17] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology 64 (2015) 1–18.
[18] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, 2007.
[19] D. Qiu, B. Li, S. Ji, H. Leung, Regression testing of web service: A systematic mapping study, ACM Computing Surveys (CSUR) 47 (2) (2014) 21:1–21:46.
[20] L. Gonçales, K. Farias, B.d. Silva, J. Fessler, Measuring the cognitive load of software developers: A systematic mapping study, in: Proceedings of the 27th International Conference on Program Comprehension, in: ICPC '19, IEEE Press, Piscataway, NJ, USA, 2019, pp. 42–52.
[21] S.C. Müller, Measuring software developers' perceived difficulty with biometric sensors, in: Proceedings of the 37th International Conference on Software Engineering - Volume 2, in: ICSE '15, IEEE Press, Piscataway, NJ, USA, 2015, pp. 887–890.
[22] J. Blasco, T.M. Chen, J. Tapiador, P. Peris-Lopez, A survey of wearable biometric recognition systems, ACM Computing Surveys (CSUR) 49 (3) (2016) 43:1–43:35.
[23] S.C. Muller, T. Fritz, Using (bio)metrics to predict code quality online, in: 2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE), in: ICSE '16, ACM, New York, NY, USA, 2016, pp. 452–463.
[24] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J.C. Hofmeister, A. Brechmann, Simultaneous measurement of program comprehension with fmri and eye tracking: A case study, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–10.
[25] N. Peitek, J. Siegmund, S. Apel, C. Kästner, C. Parnin, A. Bethmann, T. Leich, G. Saake, A. Brechmann, A look into programmers' heads, IEEE Transactions on Software Engineering (2018).
[26] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, A. Brechmann, Measuring neural efficiency of program comprehension, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, in: ESEC/FSE 2017, ACM, 2017, pp. 140–150.
[27] K. Figl, Comprehension of procedural visual business process models, Business and Information Systems Engineering (ISSN: 1867-0202) 59 (1) (2017) 41–67.
[28] U. Obaidellah, M. Al Haek, P.C.-H. Cheng, A survey on the usage of eye-tracking in computer programming, ACM Computing Surveys (CSUR) 51 (1) (2018) 1–58.
[29] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, A systematic literature review on the usage of eye-tracking in software engineering, Information and Software Technology 67 (2015) 79–107.
[30] K.S. Davies, Formulating the evidence based practice question: a review of the frameworks, Evidence Based Library and Information Practice 6 (2) (2011) 75–80.
[31] X. Huang, J. Lin, D. Demner-Fushman, Evaluation of PICO as a knowledge representation for clinical questions, in: AMIA Annual Symposium Proceedings, Vol. 2006, American Medical Informatics Association, 2006, p. 359.
[32] B.A. Kitchenham, E. Mendes, G.H. Travassos, Cross versus within-company cost estimation studies: A systematic review, IEEE Transactions on Software Engineering 33 (5) (2007) 316–329.
[33] S.C. Müller, T. Fritz, Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress, in: IEEE/ACM 37th IEEE International Conference on Software Engineering, in: ICSE '15, 2015, pp. 688–699.
[34] J.L. Plass, S. Kalyuga, Four ways of considering emotion in cognitive load theory, Educational Psychology Review (2019) 1–21.
[35] L. Chen, M.A. Babar, H. Zhang, Towards an evidence-based understanding of electronic data sources, in: 14th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2010, pp. 1–4.
[36] N. Peitek, S. Apel, A. Brechmann, C. Parnin, J. Siegmund, CodersmUSe: multi-modal data exploration of program-comprehension experiments, in: Proceedings of the 27th International Conference on Program Comprehension, IEEE Press, 2019, pp. 126–129.
[37] J. Castelhano, I.C. Duarte, C. Ferreira, J. Duraes, H. Madeira, M. Castelo-Branco, The role of the insula in intuitive expert bug detection in computer code: an fMRI study, Brain imaging and behavior 13 (3) (2019) 623–637.
[38] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in software engineering, Springer Science & Business Media, 2012.
[39] J.L. Fleiss, Measuring nominal scale agreement among many raters., Psychological bulletin 76 (5) (1971) 378.
[40] J.R. Landis, G.G. Koch, The measurement of observer agreement for categorical data, Biometrics (1977) 159–174.
[41] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: a critical review and guidelines, in: Proceedings of the 38th International Conference on Software Engineerin, 2016, pp. 120–131.
[42] J. Sweller, Element interactivity and intrinsic, extraneous, and germane cognitive load, Educational psychology review (ISSN: 1573-336X) 22 (2) (2010) 123–138.
[43] F.G. Paas, J.J. Van Merriënboer, Instructional control of cognitive load in the training of complex cognitive tasks, Educational psychology review 6 (4) (1994) 351–371.
[44] F. Lotte, L. Bougrain, A. Cichocki, M. Clerc, M. Congedo, A. Rakotomamonjy, F. Yger, A review of classification algorithms for EEG-based brain–computer interfaces: a 10 year update, Journal of neural engineering 15 (3) (2018) 031005.
[45] M.X. Cohen, Where does EEG come from and what does it mean? Trends in neurosciences 40 (4) (2017) 208–218.
[46] S. Radevski, H. Hata, K. Matsumoto, Real-time monitoring of neural state in assessing and improving software developers' productivity, in: International Workshop on Cooperative and Human Aspects of Software Engineering, in: CHASE '15, Piscataway, NJ, USA, 2015, pp. 93–96.
[47] S. Debener, M. Ullsperger, M. Siegel, K. Fiehler, D.Y. Von Cramon, A.K. Engel, Trial-by-trial coupling of concurrent electroencephalogram and functional magnetic resonance imaging identifies the dynamics of performance monitoring, Journal of Neuroscience 25 (50) (2005) 11730–11737.
[48] D. Fucci, D. Girardi, N. Novielli, L. Quaranta, F. Lanubile, A replication study on code comprehension and expertise using lightweight biometric sensors, in: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), IEEE, 2019, pp. 311–322.
[49] E. Soloway, K. Ehrlich, Empirical studies of programming knowledge, IEEE Transactions on software engineering (5) (1984) 595–609.
[50] B. Floyd, T. Santander, W. Weimer, Decoding the representation of code in the brain: an fMRI study of code review and expertise, in: 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 175–186.
[51] R. Wieringa, N. Maiden, N. Mead, C. Rolland, Requirements engineering paper classification and evaluation criteria: A proposal and a discussion, Requirements engineering 11 (1) (2005) 102–107.
[52] Mindwave mobile 2, 2021, http://neurosky.com/biosensors/eeg-sensor/ accessed on 23 Feb. 2021.
[53] Emotiv epoc+, 2021, https://www.emotiv.com/epoc/ accessed on 23 January 2021.

[54] M. Zimoch, R. Pryss, J. Schobel, M. Reichert, Eye tracking experiments on process model comprehension: lessons learned, in: Enterprise, Business-Process and Information Systems Modeling, Springer, 2017, pp. 153–168.

[55] A. Duraisingam, R. Palaniappan, S. Andrews, Cognitive task difficulty analysis using EEG and data mining, in: 2017 Conference on Emerging Devices and Smart Systems, IEEE, 2017, pp. 52–57.

[56] J. Siegmund, N. Siegmund, S. Apel, Views on internal and external validity in empirical software engineering, in: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Vol. 1, IEEE, 2015, pp. 9–19.

[57] J. Siegmund, Program comprehension: Past, present, and future, in: 23rd International Conference on Software Analysis, Evolution, and Reengineering, Vol. 5, IEEE, 2016, pp. 13–20.

[58] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: Asia-Pacific Software Engineering Conference, 2016, pp. 153–160.