International Journal of Computer Applications (0975 - 8887)
Volume 177 - No.34, January 2020

Towards the Measurement of Mental Effort in Software Engineering: A Research Agenda

Lucian Gonçales
Applied Computing Graduate Program
University of Vale do Rio dos Sinos

Kleinner Farias
Applied Computing Graduate Program
University of Vale do Rio dos Sinos

ABSTRACT

Cognitive load refers to the mental effort applied to perform cognitive processes. In software engineering, developers are involved in cognitive processes such as program comprehension and change tasks. Measuring cognitive load would be a human-centered solution, instead of using measurements based on artifacts which have been shown to have no correlation with developers' perception. Therefore, evaluate the cognitive load of the developer has potential to leverage the identification of source code issues and also improve the developers experience with their work environment. To determine a potential searcher to identify and organize this article a research agenda in relation to the measure of cognitive load of developers. This article also discusses the implications of using the cognitive load as a multipurpose indicator in software engineering. Finally, this article provides for practitioners and researchers a way to advance in the research about developers' cognitive load in software engineering in realistic scenarios.

General Terms

Software engineering, Human-factor

Keywords

Cognitive Load, Program Comprehension, Source code, Research Agenda

1. INTRODUCTION

The mental effort applied to resolve a task is refered to cognitive load [22][19]. In software engineering, the developers' cognitive load would be a potential indicator with more predictive capabilities of software quality or defects [21][5][7]. This because, the software engineering tasks are of human nature [15] [17] [4], and traditional metrics, such as, cyclomatic complexity, and line of codes can only point defects or indicate about some inconsistency in the project only after the changes were effectively made [19][17]. Furthermore, the cognitive load indicators have been used for many purposes in research of software engineering. Igor Crk 2016 [6] measured the cognitive load to classify and categorize the level of program comprehension of software developers. Sebastian Muller [19] used machine learning techniques that used cognitive load indicators to point quality concerns in a source code. Developers' cognitive load was used also to estimate the task difficulty. The Cognitive load of developers could be an important indicator, being the human-centric measures that was missing in tasks that is constantly affected by the human factors.

Therefore, indicators based on cognitive load have a emerging potential to support several tasks in software engineering. For this, researchers and practitioners are interested in the next steps to advance in the research and application of cognitive load in software engineering. The further steps could be delineated in form of research challenges, and possible implications of adopting this indicator. The related literature investigated the cognitive load for biometric recognition purposes, focused the analysis of shortcomings, and advantages on their sensors [3], the cognitive biases on software engineering [18], and investigated the cognitive workload on personal health records [23]. Thus, the previous studies did not focus on further research steps and implications of estimating the cognitive load in software engineering.

This work aims to identify the challenges and implications of cognitive load in software engineering, forming then, further directions for this field of research. Academia will benefit from this work because researchers could use the challenges in this work to inspire their research and advance in relation to open challenges. The practitioners of industry may use this study to guide their commercial interests, and reflect on the consequences of adopting such measures in their daily work. Furthermore, this work also listed the possible implications on adopting this measure, among them, the ethical and practical consequences.

For this, this study conducted a scoping review, based on well-established guidelines in the literature [20]. In particular, this work was based on procedures of systematic reviews because they provide search protocols. Therefore, the works that we extracted the challenges and implications were selected based on criteria. Initially, 807 potential studies were retrieved from two widely-know search engines, and 14 studies were selected after an insightful process of selection. From these studies was identified further challenges and implications of cognitive load in software engineering.

The remaining of this paper consists on the following. Section 2 presents the basic concepts regarding cognitive load in software engineering. Section 3 describes the methodology. Section 4 presents the future challenges regarding cognitive load in software engineering. Section 5 presents the possible implications of adopting cognitive load in software engineering. Section 6 relates this work to the existing literature. Section 7 describes the final remarks of this work.

2. COGNITIVE LOAD

Figure 1 presents the ecosystem that comprises the measurement of cognitive load in software engineering. In general, in software engineering, this ecosystem comprises on a process with are composed by four three main steps [19][15] [17]: (i) retrieve raw signals from users from a device or a set of devices (s); (ii) the raw signals from sensors are treated to remove noise, then, a cognitive load related metric or a set of metrics are applied on treated signals. The goal of this step is to extract and define a valuable set of metrics or features from the raw signals (f(r) $\rightarrow$ v); (iii) these set of features or measures are analyzed on machine learning techniques aiming to correlate or classify to an outcome measure, such as, the level of productivity on a task, and the estimation of being stuck. Therefore, the measures of cognitive load are usually associated with the classification and prediction problems.

Biometric Devices: Several devices can be used to capture signals related to cognitive load [19, 3]. Including the electroencephalogram (EEG) which retrieve electrical signals from the brain, the fNIRS which capture the blood flow from the brain, and eye tracking, device which tracks the eye movements. Another sensor that could be used to collect cognitive load is a wristband that captures information such as heart beating, and respiratory rate, information's which are related to cognitive load.

Cognitive load: The cognitive load is obtained through the reflection of the signals provided by the previously devices [6]. While users are concentrated in a task, the devices captures the manifestations of their body. For instance, during a task that demands high effort, the heart rate and respiratory rate increases. Also the electrical signals alters, including an evident desynchronization of brain signals.

Applications: There are a lot of applications which researchers in the software engineering research field are interested such as, identifying the level of expertise of developers [6], measuring the task difficulty according to developers' perception [19, 14], and improving the developers' well-being [10]. However, there are many shortcomings which academia and industry must overcome in order to deploy the cognitive load indicator in realistic scenarios definitively. This work is focused to address this problem.

3. METHODOLOGY

This section defines the methodology used to retrieve the state-of-art literature about the measures of cognitive load in software engineering. From this selected literature, we derived the further challenges and implications of applying the cognitive load in software engineering. The methodology of this work consists in a scoping review, which we seek to systematically selected related works based on criteria of selection. Section 3.1 presents the objective and research questions of this work. Section 3.2 presents the strategy to select the potential studies. Section 3.3 the process of studies selection.

3.1 Objective and Research Questions

This study has two main objectives: (1) to grasp a research agenda in relation to the cognitive load measures in software engineering. (2) List which implications that industry suffers after industry adopting cognitive load. For this, this study comprises in two research questions: (1) What are the main challenges about the measures of cognitive load in software engineering? (2) What are the implications on dealing of these measures?

To answer these questions, a scoping review was conducted to select systematically the related literature.

3.2 Search Strategy

This section presents the method for strategy used to search studies in the literature. The strategy consisted on building a search string, define the criteria for study selection, and define a main search engine to conduct the search. This search strategy was built based on well-defined guidelines in the literature [20].

Search String. The search string was built based following the PIO method (Population, Intervention, and Outcome). The populations group is in respect of sensors used to collect cognitive load ("Brain-Computer Interfaces"). The intervention group means the measures taken from the populations ("cognitive load"). The outcomes group refer to the where contributions about the relation of population and intervention referred ("Software engineering"). The keywords were extracted from main references to this research field. The search string bellow was formed after relating the strings of the different categories with the Boolean AND. The result is the string bellow.

"Brain-Computer Interfaces" AND "cognitive load" AND "Software engineering"

Selection Criteria. We defined criteria for inclusion and exclusion of studies. These criteria are described bellow:

EC1- Grey literature: This criterion removed studies that fit into a category "Grey literature", which consists on patent registration, or documents describing "call for papers"';

EC2- Title and Abstract- This criterion removed studies by title and abstract, which not presents any relation to the field of research of this work;

EC3- Language: This criterion removed studies by analyzing their native language. English language is the default language in this study. Therefore, papers not written in English were removed;

EC4- Semantic: This criteria discard studies, which contain some terms of the search string, but the study did not address issues of the research field (cognitive load in software engineering) by analyzing their title and abstract;

EC5- Duplicate: This criterion removed duplicate studies;

Search Engine. In this work, Google Scholar and Scopus was the search engines used to conduct the selection process. These engines were selected because they cover works published in main journals and conferences related to computer science, which contain works investigating cognitive load in software engineering.

3.3 Study Selection

Figure 2 shows that the process for selection of studies. In the first step, the defined search string was applied on the Google Scholar and Scopus search engines. A total of 807 works were found in the search engines. In Google Scholar were obtained 755 works, and 52 studies were returned by Scopus' search engine. Next, the articles that registered repeated records were filtered. A total of 97 articles were removed because they were in duplicate (Applying EC5 criteria), and thus resulting in 710 articles to analyze. After, it was verified whether within this 710 articles sample contained grey literature, and which among these articles were inside the research scope of this work. Thus, 87 works were removed because they were not elaborated in English (EC3) or consisted in grey literature materials (EC1). Moreover, 559 studies were removed after analyzing their title and abstract. These works were removed because they were not within the research scope based on information in their title and abstract (EC2). For the next step, remained a total of 64 articles, which were fully analyzed. After a full-text checking, 50 works were discarded in the final list. These works were removed because they were short papers, or definitely were not aligned to the

purposes of this article. Finally, 14 works composed the final list of representative works in Appendix A. These works were assigned with identifiers that varies from W01 until W14 representing the selected 14 works in the filtering process respectively. The challenges list that we identified are related to these identifiers respectively.
This final list of selected articles was used to answer our defined research questions of this work: (1) delineate the future challenges and (2) future implications of measuring cognitive load in software engineering.

4. RESEARCH AGENDA AND CHALLENGES

(1) An approach to estimate the developers' cognitive workload level [W01] [W02] [W10]. Some studies found that the increasing cognitive load impacts on developers' performance during the cognitive process, such as, comprehending source code[6, 19]. In the case of certain level of cognitive workload jeopardize the developers' comprehension, then the software project probably will be compromised too [14]. The cognitive workload level also is an information that can be used to a wide range of purposes such as, for indicating the level of learning, to promote task recommendations based on their cognitive work load.
However, some challenges must be overcome in order to estimate the cognitive workload of developers precisely. For instance, it's already known that various physiological manifestations are related to cognitive workload, such as, brain waves, pupil size, and respiratory rate. Therefore, the role of researchers is to investigate which from these biometric features are really effective in estimating the cognitive workload in software engineering context. Finally, another fact about cognitive load, is that it is affected by the level of experience, and besides that, it can vary across individuals. For instance, Lee 2017 [14] controlled the cognitive load in relation to two experience levels (experienced and no experienced), while in [6] analyzed the cognitive load in relation to 5 categories of experience (from sophomore to seniors one). Therefore, the question is, which are the appropriated categories to analyze the cognitive workload?
(2) Combining EEG and Eye tracking [W01] [W14]. Capturing biometric data through a single sensor enabled a limited analysis of cognitive load during coding tasks. While the electrical signals captured by the electroencephalogram help in exploring cognitive processes such as code comprehension from the brainwaves perspective, the eye tracker data can help to locate which part of the software artifact the developer was focused on. Eye Tracking can also provides support to developers on detecting the strategy the programmer uses to manipulate the artifact. Therefore, the combination of these two sensors would make it possible to cross cognitive process data, as well as, where in the artifact, the developer dedicates their cognitive load. This would enable systematic analysis and conclusions of how the developer understands software artifacts.
Despite these advantages, there are some open questions to turn the combination of EEG and Eye Tracking working effectively. For instance, in order to conduct a coherent cognitive load analysis, the data transmission of the sensors must be synchronized. Synchronizing these data are a challenging task as both devices have characteristics regarding temporal accuracy. Time accuracy is the frequency with which data is captured over a period of time. Another challenge is how to report EEG and Eye tracking data to the developers. These devices have different characteristics and report data in different way. The way data are presented is crucial to understanding what happens during the software tasks. Another question is regarding data processing. Data processing consists of obtaining the data that is actually related to the event of interest. For example, various types of noise may reflect electroencephalogram data, such as involuntary muscle movements. These information must be filtered and removed in the processing step. The characteristics of noise types differ from sensor to sensor. Thus, specific processing for each sensor should be applied. Therefore, methods for processing data from these devices synchronously must be defined.
(3) Improve spatial resolution of data collected through the electroencephalogram [W06]. The EEG spatial resolution is an important factor in analyzes of brainwaves signals covering the maximum as possible the scalp area. An higher spatial resolution would benefit researchers to analyze precisely the source of cognitive effort on software engineering tasks. Thus, the quality of resolution depends on the number of channels the EEG device contains. These channels are those deployed on the users' scalp. More spatial resolution is obtained when more channels are deployed on the scalp, thus resulting on a higher coverage of the space of the scalp, therefore, improving the spatial resolution.
However, obtaining high resolution data is a challenging task due to the limitations on research project income: acquiring such equipments are very expensive. The consequence is that research usually adopts EEG devices with low resolution, i.e., devices with one or 14 channels such as Neurosky [12] and Emotiv [8] respectively. To overcome this problem, software engineering labs must establish partnership with specialized neuroscience labs. It's also desirable that the industry resolves this problem on producing cheaper wearable and off-the-shelf devices with an increased spatial resolution in relation to the current state-of-art devices.
(4) Define a method to filter the EEG data [W09]. Another challenge is filtering the noise of sensors' signals. Filtering the EEG data is an important step to maintain the quality of EEG data. EEG data are noisy due to power-line interferences, and muscle movements such as, eyes' movements, which constitute of artifacts that affect the analysis of the study purposes. This is because the main aim of the software engineering studies is to analyze the cognitive process of developers, such as task difficulty. Removing noisy signal is essential to analyze the data that really are related to the object of interest to the research analysis. Thus, researchers and practitioners must isolate the data that are really related to the analysis. For this, a number of techniques for filtering data exists, such as high-pass filter, low-pass filter, band-pass filter, RC offsets, etc [16]. However, each one serves to a different purpose, and can be suitable for specific situations. For example, low-pass filtering is indicated for attenuating eye movements' artifacts. Moreover, users can define specific cut-offs for each filter, i.e., the bounds which EEG signals are filtered.
Despite the existence of well-established techniques for filtering and clean the EEG Data, its not clear yet for practitioners and scientists what specific filter must be applied. For this, an important research track is to develop a framework for filtering the EEG data according to the purpose of experiment. A framework would be used by researchers as a guide to apply the appropriate filter method according to the experiment or aims of their research analysis. In other words, scientists and practitioners must be aware of what kind of filter they must use for their analysis. For this, the challenging task is to define clear guidelines for what steps and which filter to apply. Another challenge is to develop an approach for apply automatically the ideal filtering according to the behavior of EEG signal.
(5) Investigate the relation of developers' cognitive load level and Bad Smells [W03][W04]. Bad smells is a well-know list of patterns that describes source code problems [9]. Specifically, these code problems jeopardize the maintainability, and understanding of

the system. At first, it is assumed that with a low level of cognitive load, the developer is prone to inserting Bad Smells in the source code. This is because it is hypothesized that the programmer is applying mental effort to develop a more sophisticated solution in the source code. And with that, the developer can insert a range of code problems, among them, the best known are code duplication, when there are similar codes in the same line of code, a "long" class, when a class has an big size and concentrates a lot of variables and methods, and a long list of parameters in the method.

It is well-known in the literature that the presence of bad smells impairs system maintenance as well as developer understanding [13][9]. For this, a future challenge is to analyze whether certain cognitive load levels are correlated with specific bad smells. Thus, it would enable the deduction of such cognitive load level indicates a possible specific bad smell emerging in a source code. In addition, another more difficult problem would be to identify which level of cognitive load would lead to turn developers prone to insert bad smells. With these information, it would be possible to inform the developer what kind of bad smell he would be putting in the source code online. In case of it be proven, they would help developers maintain the system avoiding inserting such code smells.

(6) Analysis of the relation between developers' cognitive load and their expertise [W02] [W05] [W07] . The level of expertise influences the developers' performance on programming tasks. Studies have been relating the cognitive load level with developers' expertise level [6][14]. Studies already pointed a relation between cognitive load and their expertise level, and they concluded that expert developers apply less effort to resolve a task. Identifying the expertise of developers is a relevant information for many purposes, for instance, for identifying the level of education of the software developer. Moreover, it would support the identification of the expertise of developers in specific languages and kind of tasks. Thus, this information would be important in a realistic scenario, to recommend tasks for developers improving their skills, or to allocate them in a task which is coherent to their expertise level.

The first problem is to replicate the already existent studies, to reinforce the validation of expertise level and cognitive load level. For example, Igor Crk [6] used the EEG to measure the cognitive load of software developers, which pointed a low predictive power for developers' expertise using alpha band in tasks' correctness. Thus, other tests must be concentrated on other bands such as gamma, beta or theta for the purposed of investigating the real relation of the specific brain frequency bands and expertise level. A second challenge, among several paradigms already used to measure the developers' expertise, such as, fMRI, and EEG, further studies must evaluate the trade-offs of using these devices to analyze the developers' expertise, and how their information can be integrated in order to improve the accuracy of developers' expertise.

(7) Prediction of code quality concerns based on developers' cognitive load. [W06] Quality concerns are usually detected by traditional metrics such as cyclomatic complexity, and modular level. What that metrics has in common is that both are based on software artefacts. Thus, these metrics only can be analyzed only right after the code was modified [19]. Furthermore, literature already evidenced that artifact-based metrics are not related to the prediction of program comprehension. One of the reasons is that they are not sensitive to the developers' sensing. A metric based on developers' perspective, using their cognitive load, would be important to point the propensity of the generation of quality concerns or errors in the source code. Despite some studies already pointed that the usage of cognitive load outperformed the usual metrics, some challenges must be resolved to advance in this direction.

First, current studies are limited to relate the mental effort to a few sets of quality concerns, such as, bad comments, and bugs [19]. Further studies could discover the relation of cognitive load to improve the granularity of quality concerns. Second, they evaluated the quality concerns predictability based on classifier such as Support Vector Machines, Naïve Bayes, and Random Forrest. Consequently, they reached a good, but improvable, 40% of accuracy for classifying quality concerns. Thus, other techniques machine learning techniques could be evaluated. Third, studies predicted quality concerns based on electrodermal activity, respiratory rate, and heart beating sensors. Thus, the previous investigations neglected the usage of others widely used sensors in this kind of research such as, EEG, fMRI, and fNIRS. Using these sensors could improve the predictability power of quality concerns.

(8) Effects of emotions on developers' programming performance [W13] [W11]. During software development, the emotions of software developers vary from negative to positive emotions. Negative emotions are related to frustration, or sadness, while positive one is related to satisfaction, and happiness. Furthermore, many factors can influence the emotions of software developers, such as, time pressure, sense of progress, and the work environment. These factors influences the emotions of software developers, and consequently, their progress at work. The researchers assume that negative emotions have a negative impact on developers sense of progress, and consequently, a negative impact on their performance. In previous works, biometric data were correlated to a certain range of emotions, which they pointed that negative emotions are related to lack of progress. This data is important, because based on the developers' emotions would enable to make suggestions to the developers take a break, or execute an easier task.

However, some improvements are required to definitively related the emotions with programming performance. First, studies did not investigate the relation of cognitive load level with the wide-range of possible emotions of the developers. An investigation like this would relate whether developers experience low or high cognitive load during negative emotions. Second, future studies should improve the identification of negative and positive emotions based on biometric data. Despite previous work [W11][W12] classified successfully negative and positive emotions with high accuracy (70%) it could be improved using brain-centric data, instead of applying the respiratory rate, and electrodermal activity.

5. IMPLICATIONS ON REALISTIC SCENARIOS

(1) Promote the welfare in the software development scenario. As the cognitive load is a data that are owned by the developer, we believe these data should be used for the benefit of developers. Therefore, the best possible use of this data would imply improving the developer's welfare. Thus, indicators of cognitive load, along with indicators of emotions, should be used to identify stress, anxiety, and based on this information recommend activities outside the scope of work such as exercise, meditation, or reducing the working time. Moreover, decreasing working time has been shown to be effective in increasing productivity. Therefore, promoting developer welfare would be a promising implication because there is evidence that welfare provides a better work environment.

(2) Improved quality of software systems . With the use of cognitive load the quality of software systems is expected to improve, and software system customers will enjoy a system with fewer bugs and execution problems consequently. This is because current metrics only capture issues after changes have been made. With the use of cognitive load, it is expected to detect stress points of the developer who may lead to loss of system quality. That way, the

programmer may receive feedback from the system for him asking support from another team member or to indicate to the developer that something might go wrong with the task if he continues the activity. Therefore, preventing code quality from being compromised before code is sent to the system repository is one of the expected implications of using cognitive loading in software engineering [19].

(3) An improved estimator for indicating the presence of quality concerns. There are many software design metrics being used by analysts to demonstrate the presence of code quality concerns. Metrics such as the number of lines, number of methods, number of parameters, number of global attributes, among others, may indicate source code problems. These problems could be an architectural standard that were not followed by developers, such as denying following modular specifications of features. However, the notion of size is not accurate in software engineering. The number of methods, or lines considered to be large, or small, is not accurate and may vary across existing projects, turning the identification of code problems inaccurate. Thus, cognitive load may imply in a developer-based indicator that indicates code problems according to developers perception. This would imply in an improved estimator that may be more accurate for indicating the presence of quality concerns.

(4) An Integrated Development Environment (IDE) responsive to the developers' cognitive load level. Another implication that is still expected by the existing literature is an IDE that is aware in relation of the developers' perception. In addition, it is believed that if a better effectiveness of cognitive load as an indicator of software quality is proven, it may signal the developer's perception of the current activity. Based on this indicator, the IDE could indicate which tasks the developer might perform. Based on the cognitive load the IDE could adjust the screen layout such as source code size or colors that should be presented to the user. However, before that, a range of experiments is expected to correlate which tasks should be recommended to developers. It is at least estimated that for each developer, a unique configuration should be established between recommendation types.

(5) Ethical and privacy aspects. Data related to cognitive load, obtained through biometric sensors such as heart rate, electroencephalogram, and eye tracker, are private data from the developers. Therefore, companies should be concerned about maintaining the confidentiality of information. It would be a very unethical action to use contract clauses to enable the sale of the biometric data of its employees. Therefore, companies should behave ethically and keep data confidential. For instance, a company should not maintain proprietary rights to such data. In addition, the literature also points out that using this data to evaluate, promote or recruit staff would be unprofessional. Qualifying a person for the quality of their physical attributes is a serious ethical problem [6].

6. RELATED WORKS

Gui et al. 2019 [11] conducted a survey about brain biometrics, i.e., aspects of the brain for identification of users in general. Author argues that the electrical brain signals provide a potential and important aspect to identify users because they are unique and exclusive to each user. Thus, according to authors EEG is a signal that provides confidentiality. The authors also argue that multimodal recognition system would be a great future challenge is a solution to verify the identity by various sources of signals (eye, heart) strengthening the identification of users. The authors also pointed keeping the stability of EEG signal is a challenging task, because it can be easily affected by user emotions.

Bablani et al. 2019 [1] conducted a survey about techniques that can make an interface between brain and machines. This study is not focused on software engineering, and refers to brain-centric devices in a general way. The authors also highlighted the methods to enable to transform the brain waves signal in a readable information to machines. For this, they highlighted methods such as the Convolutional Neural Network (CNN) that extracts information from brain autonomously. These methods are also important in a software engineering context, because these devices are deployed inside an environment which measures the cognitive load of developers based on their biometric indicators.

Wilbanks and McMullan 2018 [23] reviewed the literature about the application of cognitive workload of Electronic Health Records (EHR) users. Electronic Health Records provides a decentralized manner to clinicians to access the patients' record, and historical information about their deceases. Authors argued that high cognitive workload can be interpreted as erroneously interpreted information's of clinicians from EHR's. The bad design and inadequate usage of EHR leverage the changes of them make mistakes implying the increasing of death rates. This work is important for software engineering, because it highlights the importance which cognitive load can play in the design and project of users' interfaces, such as, in IDE's.

Lotte 2007 [16] presents and describes the machine learning techniques for classifying data obtained from the electroencephalogram. Authors focused on challenges of extracting classes from brain-centric data as well as choose the appropriated machine learning technique for specific cases. Lotte also proposed a guideline to schematize which technique developers must adopt. This work is very important because it points which technique is best suited for classifying brainwaves data. Authors pointed that the support vector machine is the most appropriated one. Authors also highlighted that the performance of the algorithms must be improved.

Mohanani et al. 2018 [18] systematically listed the cognitive biases of developers during software-development tasks. The authors argue that cognitive biases such as developers' overconfidence, and cognitive bias on decisions could jeopardize the software quality. It was also highlighted the need for producing techniques to manage and avoid cognitive biases during software development. It would avoid that personal biases of collaborators on affecting the software purpose. The authors highlighted the sense of cognitive bias could be incorporated in learning environments. Authors also highlighted that the human-factor influences in software quality significantly rather than measures from software artifacts.

Bateson et al. [2] developed a scheme to categorize the brain-computer interfaces. In particular, they classified that electroencephalogram off-the-shelf, and that vendors usually pointed as "mobile" and "versatile" devices. However, the authors verified these devices in detail and found limitations in these sensors. For instance, they did not provide much mobility as argued by vendors and for this, they categorized these devices according to their real characteristics. This work makes evident that the devices are one of the parts of the problem that must be overcome in order to enable the definitive adoption of these devices in real environments.

7. CONCLUSION

Cognitive load has the potential to indicate the welfare of developers and to improve the quality of work of developers. Whereas, the literature has already demonstrated the effectiveness of this metric to indicate developers' level of expertise, as well as to demonstrate the level of difficulty perceived by developers. Therefore, it is in the interest of academia and industry to have a research direction

to explore the benefits of adopting developers' cognitive load in software-development tasks. Therefore, the objective of this work was to complement the existing literature and provide future directions to researchers and practitioners about the application of cognitive load of software developers.

Overall, the challenges shows that cognitive load has important steps to advance toward a effective indicator to be adopted in software engineering. The challenges also reinforced what was found in the literature, i.e., that the cognitive load is a kind of measure that can used for many purposes, such as, measuring the quality of source code, until for the identification of developers expertise. In fact, the cognitive load of developers based on biometric measurements has potential to be an indicator for effectively integrating the human-factor in software development environment. Moreover, the implications of the adoption of cognitive load in realistic scenarios shows that the industry and academics would benefit mainly from an integrated IDE that adapts to the cognitive demands of software developers. On other hand, the usage of cognitive load implies on development respecting privacy and ethical aspects regarding the data obtained from developers.

Acknowledgment

This study was financed in part by the Conselho Nacional de Desenvolvimento Cientfico e Tecnolgico (CNPq) e Coordenao de Aperfeioamento de Pessoal de Nvel Superior - Brasil (CAPES) - Finance Code 001.

Appendix A - Selected Studies

W01. T. Fritz and S. C. Müller. Leveraging biometric data to boost software developer productivity. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 5, pages 6677, March 2016.

W02. Igor Crk and Timothy 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), pages 46014604, Aug 2016.

W03. Randall K. Minas, Rick Kazman, and Ewan Tempero. Neurophysiological impact of software design processes on software developers. In Augmented Cognition. Enhancing Cognition and Behavior in Complex Human Environments, pages 5664. Springer International Publishing, 2017.

W04. Sebastian C. Müller. Measuring software developers' perceived difficulty with biometric sensors. In Proceedings of the 37th International Conference on Software Engineering - Volume 2, ICSE 15, pages 887890, Piscataway, NJ, USA, 2015. IEEE Press.

W05. Erik W Anderson, Kristin C Potter, Laura E Matzen, Jason F Shepherd, Gilbert A Preston, and Cláudio T Silva. A user study of visualization effectiveness using EEG and cognitive load. In Computer Graphics Forum, volume 30, pages 791800. Wiley Online Library, 2011.

W06. Federico Cirett Galán and Carole R. Beal. EEG estimates of engagement and cognitive workload predict math problem solving outcomes. In Proceedings of the 20th International Conference on User Modeling, Adaptation, and Personalization, UMAP12, pages 5162, Berlin, Heidelberg, 2012. Springer-Verlag.

W07. Yueran Yuan, Kai-min Chang, Jessica Nelson Taylor, and Jack Mostow. Toward unobtrusive measurement of reading comprehension using low-cost EEG. In Proceedings of the Fourth International Conference on Learning Analytics And Knowledge, LAK 14, pages 5458, New York, NY, USA, 2014. ACM.

W08. Andre N Meyer, Thomas Zimmermann, and Thomas Fritz. Characterizing software developers by perceptions of productivity. In Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pages 105110. IEEE Press, 2017.

W09. Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. Real-time monitoring of neural state in assessing and improving software developers' productivity. In Proceedings of the Eighth International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 15, pages 9396, Piscataway, NJ, USA, 2015. IEEE Press.

W10. Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. Measuring neural efficiency of program comprehension. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, pages 140150. ACM, 2017.

W11. A. R. Subhani,W. Mumtaz, M. N. B. M. Saad, N. Kamel, and A. S. Malik. Machine learning framework for the detection of mental stress at multiple levels. IEEE Access, 5:1354513556, 2017.

W12. Sebastian C. Müller and Thomas Fritz. Stuck and frustrated or in flow and happy: Sensing developers emotions and progress. In Proceedings of the 37th International Conference on Software Engineering - Volume 1, ICSE 15, pages 688699. IEEE Press, 2015.

W13. - N. Jatupaiboon, S. Pan-ngum, and P. Israsena. Emotion classification using minimal EEG channels and frequency bands. In The 2013 10th International Joint Conference on Computer Science and Software Engineering (JCSSE), pages 2124, May 2013.

W14. - Manuela Züger and Thomas Fritz. Interruptibility of software developers and its prediction using psycho-physiological sensors. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 15, pages 29812990, New York, NY, USA, 2015. ACM.

8. REFERENCES

[1] Annushree Bablani, Damodar Reddy Edla, Diwakar Tripathi, and Ramalingaswamy Cheruku. Survey on brain-computer interface: An emerging computational intelligence paradigm. *ACM Comput. Surv.*, 52(1):20:1–20:32, February 2019.

[2] Anthony D Bateson, Heidi A Baseler, Kevin S Paulson, Fayyaz Ahmed, and Aziz UR Asghar. Categorisation of mobile eeg: A researchers perspective. *BioMed research international*, 2017, 2017.

[3] Jorge Blasco, Thomas M. Chen, Juan Tapiador, and Pedro Peris-Lopez. A survey of wearable biometric recognition systems. *ACM Comput. Surv.*, 49(3):43:1–43:35, September 2016.

[4] Daniel Cernea, Peter-Scott Olech, Achim Ebert, and Andreas Kerren. Measuring subjectivity. *KI-Künstliche Intelligenz*, 26(2):177–182, 2012.

[5] Celia Chen, Reem Alfayez, Kamonphop Srisopha, Lin Shi, and Barry Boehm. Evaluating human-assessed software maintainability metrics. In *National Software Application Conference*, pages 120–132. Springer, 2016.

[6] Igor Crk and Timothy 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)*, pages 4601–4604, Aug 2016.

[7] Jonathan Dorn. A general software readability model. *MCS Thesis available from (http://www. cs. virginia. edu/˜ weimer/students/dorn-mcs-paper. pdf)*, 2012.

[8] Emotiv. Testbench specifications, emotiv, 2014.

[9] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[10] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. What happens when software developers are (un) happy. *Journal of Systems and Software*, 140:32–47, 2018.

[11] Qiong Gui, Maria V. Ruiz-Blondet, Sarah Laszlo, and Zhanpeng Jin. A survey on brain biometrics. *ACM Comput. Surv.*, 51(6):112:1–112:38, February 2019.

[12] J Katona, I Farkas, T Ujbanyi, P Dukan, and A Kovari. Evaluation of the neurosky mindflex eeg headset brain waves data. In *2014 IEEE 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 91–94. IEEE, 2014.

[13] Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. Tracing software developers' eyes and interactions for change tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 202–213, 2015.

[14] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. Mining biometric data to predict programmer expertise and task difficulty. *Cluster Computing*, Jan 2017.

[15] Yisi Liu, Olga Sourina, and Minh Khoa Nguyen. Real-time eeg-based human emotion recognition and visualization. In *2010 International Conference on Cyberworlds*, pages 262–269, Oct 2010.

[16] F Lotte, M Congedo, A Lécuyer, F Lamarche, and B Arnaldi. A review of classification algorithms for EEG-based brain–computer interfaces. *Journal of Neural Engineering*, 4(2):R1–R13, jan 2007.

[17] Randall K. Minas, Rick Kazman, and Ewan Tempero. Neurophysiological impact of software design processes on software developers. In *Augmented Cognition. Enhancing Cognition and Behavior in Complex Human Environments*, pages 56–64. Springer International Publishing, 2017.

[18] R. Mohanani, I. Salman, B. Turhan, P. Rodrguez, and P. Ralph. Cognitive biases in software engineering: A systematic mapping study. *IEEE Transactions on Software Engineering*, 2018.

[19] Sebastian C. Müller. Measuring software developers' perceived difficulty with biometric sensors. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 887–890, Piscataway, NJ, USA, 2015. IEEE Press.

[20] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[21] D Srinivasulu, Adepu Sridhar, and Durga Prasad Mohapatra. Evaluation of software understandability using rough sets. In *Intelligent Computing, Networking, and Informatics*, pages 939–946. Springer, 2014.

[22] John Sweller, Jeroen JG Van Merrienboer, and Fred GWC Paas. Cognitive architecture and instructional design. *Educational psychology review*, 10(3):251–296, 1998.

[23] Bryan A Wilbanks and Susan P McMullan. A review of measuring the cognitive workload of electronic health records. *CIN: Computers, Informatics, Nursing*, 36(12):579–588, 2018.

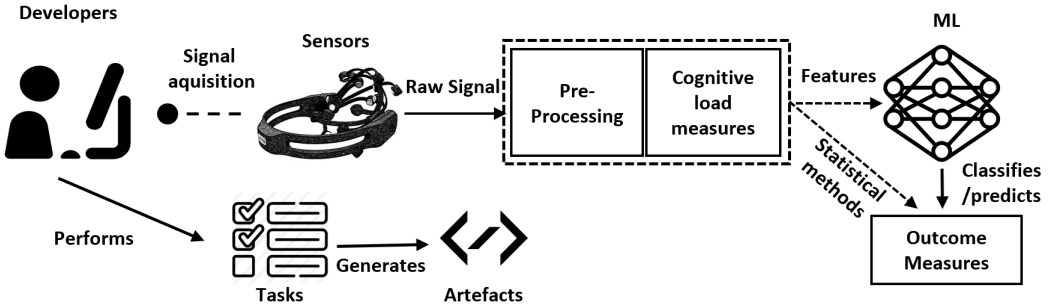

Fig. 1. General view of the ecosystem involving the measurement of developers mental effort.

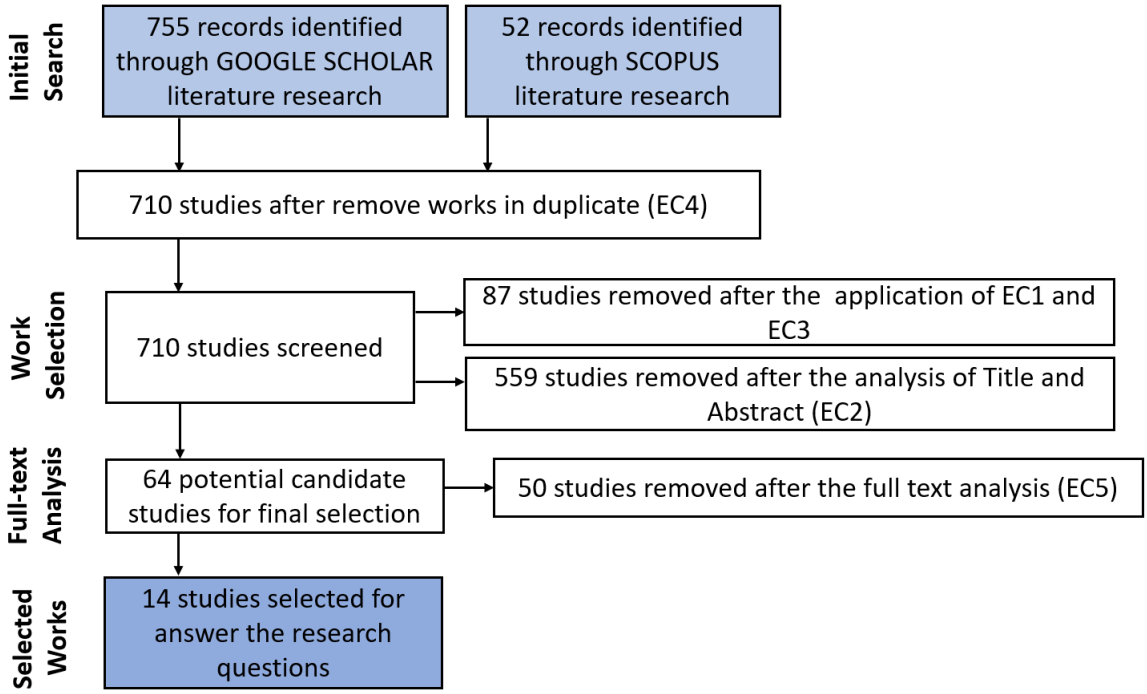

Fig. 2. Process for selection of studies.