Journal of Computer Languages 79 (2024) 101269

Contents lists available at ScienceDirect

Journal of Computer Languages

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

An empirical approach to understand the role of emotions in code comprehension

Divjot Singh *, Ashutosh Mishra, Ashutosh Aggarwal

Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala, 147004, India

ARTICLE INFO

Keywords:
Code comprehension
Systematic literature review
Emotions
Cognitive skills

ABSTRACT

Programming and cognitive skills are two pivotal abilities of programmers to maintain software products. First, this study included a systematic literature review on code comprehension, emotions, cognitive psychology, and belief-desire-intention domains to analyse various code comprehension monitoring techniques, performance metrics, and computational methodologies. Second, a case study is conducted to examine the influence of various emotional stages on programmers' programming and cognitive skills while comprehending the software code. The categorization of the participants is done empirically based on their expertism level, and the same results are verified using various machine learning models and performance metrics.

1. Introduction

Over the past few decades, there have been revolutionary shifts in the computer software and hardware industries. Constant innovations and new technologies have caused this paradigm to shift rapidly. Software failure can quickly cost companies millions or even billions of dollars, so it has become necessary for a company to keep adapting to the latest trends and requirements; hence, software is updated regularly [1]. When a software product undergoes modifications after delivery to address faults, remove bugs, enhance performance, or adapt to changes in the environment, this process is referred to as software maintenance [2]. It considered an integral part of the software development life cycle (SDLC) [3,4]. It contains various phases, including requirements engineering, architecture, design, implementation, testing, deployment, and maintenance [5]. The effectiveness of the maintenance phase is hinged upon the preceding phases, such as the implementation phase, which should produce a code that possesses qualities of readability, comprehensibility, and modifiability. The absence of structured codes and limited understanding of software among programmers can have an unfavourable impact on maintenance activities. The development of code structure and its comprehension vary according to the familiarity of programmers with domain-specific languages (DSLs) [6] or general-purpose languages (GPLs) [7]. DSLs are widely acknowledged to possess a more straightforward notation than GPLs because of their inherent nature as programming languages designed for specific domains [8] whereas GPLs are not domain-specific and can be applied to a wide variety of fields [9].

According to Ref. [10], the job of maintaining software is mentally taxing and demands high intellectual activities. Consequently, it is instructive to study how programmers regularly deal with software-related issues. Because software consists of computer programming, understanding its source code requires a wide range of technical and mental abilities. Programmers must comprehend the source code using mental processes known as higher-order skills [11]. These cognitive skills are the fundamental aspects of cognitive psychology, in which higher-order cognitive skills encompass complex cognitive processes and play a pivotal role in tasks that require advanced reasoning, problem-solving, and decision-making abilities [12].

During the process of debugging the source code, programmers may find themselves facing time constraints that can affect their emotions. Emotional stages affect programmers' cognitive and comprehension skills. Anger and irritation can affect programmers' judgment and decision-making when troubleshooting an error [13]. The comprehension process also requires the programmer to believe that the source code contains an error, and amending it will fix the problem. The knowledge gained by programmers to fix the issues represents their belief, which leads to a desire for programmers to debug codes and solve problems. The intention is their dedication, attention, and sincerity to achieve their goals or desires [14].

The motivation for our research is drawn from studies [15,16], which focus on comprehending the impact of emotions on programmers' higher-order cognitive skills, along with their programming skills.

* Corresponding author.
E-mail addresses: dsingh_phd21@thapar.edu (D. Singh), ashutosh.mishra@thapar.edu (A. Mishra), ashutosh.aggarwal@thapar.edu (A. Aggarwal).

https://doi.org/10.1016/j.cola.2024.101269
Received 22 July 2023; Received in revised form 5 February 2024; Accepted 1 March 2024
Available online 2 March 2024

Programming skills are the output of programmers, but cognitive skills define how logic has been achieved. In the process of debugging a code snippet, the level of understanding of each line of code is conditioned upon its significance, which, in turn, elicits a range of emotional responses from programmers [17,18]. Emotions play a vital role in feeling confused with frustration and happiness. If programmers have various emotional states, they will impact their debugging performance [13]; hence, it becomes crucial for programmers to stay motivated and use their programming skills to perform tasks [15,16].

In our work, we have assumed that the participating programmers are well-versed in their programming skills, so we have devoted our attention to programming and higher-order cognitive skills of the programmers and how they are affected by the different states of emotions. In this regard, an empirical study measuring the expertise level of programmers based on their cognitive load (i.e., the combination of code understanding skills and emotions) is performed, and its results are discussed in a later section.

To make this idea more intuitive, a brief systematic literature review (SLR) of existing methods for measuring cognitive skills and various performance metrics used by such approaches is presented in this paper, and the following research questions are defined and investigated.

RQ1. What monitoring methods have been used for code comprehension?

RQ2. Which performance metrics have been used in the research works?

RQ3. What are the various computational methods used in various works?

2. Background

Once the software is deployed in the global market, programmers should remain vigilant to maintain a smooth operation. Even after rigorous testing, it is possible that some bugs or loopholes remain in software related to source code [19]. Thus, it becomes important for programmers to know how to debug codes within a limited time frame. This type of maintenance is known as corrective maintenance. There are four main categories of software maintenance: adaptive, corrective, perfective, and preventive maintenance [1]. Users occasionally demand new or additional software features. Therefore, programmers try to meet these demands by performing perfective maintenance and introducing the required changes without changing the original schema of the software.

To perform maintenance tasks, programmers must use their programming and cognitive skills. Code-reading, writing, and debugging are tasks that have been used to test the cognitive skills of programmers. To identify bugs, programmers comprehend the code either line-by-line (top-down approach) or try to understand the code by analysing key identifiers and functions (bottom-up approach) [20]. After highlighting these issues, programmers use cognitive skills such as reasoning, decision-making, and problem-solving to debug and fix errors in the code. Since debugging is a time-related task [21], programmers are sometimes unable to comprehend issues within a time constraint and display a wide range of emotional states, including happiness, sadness, surprise, fear, and anger, among others [13,17,18]. Therefore, it is interesting to compare the behaviour of programmers in terms of how they perform in different emotional states while comprehending the source code [13,15]. The question arises: Will this situation impact cognitive skills?

In summary, the main focus of this work is to understand the influence of various emotional states of programmers on higher-order cognitive and source code comprehension skills while performing corrective and perfective maintenance tasks.

Several monitoring techniques have been used by programmers for source code comprehension, and these techniques are discussed in the following subsection.

2.1. Code comprehension

Code comprehension encompasses the cognitive skills required to comprehend and derive meaning from computer code, which is commonly expressed in programming languages. The process entails engaging in reading, analysing, and mentally processing the code to acquire a comprehensive comprehension of its functionality, structure, and purpose [22]. The ability to comprehend code is an essential skill of programmers, as they frequently encounter the need to engage with pre-existing codebases, grasp their underlying mechanisms, and carry out alterations or enhancements [23].

2.2. Monitoring techniques during code comprehension

Programmers comprehend several code snippets or modules and use their cognitive skills while understanding and debugging the code. It is important to understand how programmers utilize various skills such as emotions, psychology, and belief-desire-intention (BDI) for code comprehension. Researchers have used various types of monitoring techniques to understand and observe how code comprehension is performed by programmer [24–27]. Researchers have attempted to understand the workings of a programmer's brain.

- **Coding tasks**: The cognitive and programming skills of programmers are tested using various coding tasks, including code reading, code manipulation, code writing, and code debugging. The results are corroborated in interview sessions [24].
- **Electroencephalogram (EEG)-** EEG signals are electrical signals generated when neurons in the brain are activated. In this way, researchers can determine which areas of the brain become active [25].
- **Functional Magnetic Resonance Imaging (fMRI)**: What is better if a person can look inside the brain and see how it works? fMRI provides brain images to identify the parts of the brain that are activated when a cognitive or programming task is performed [26].
- **Eye tracking**: When a programmer debugs a code or tries to comprehend it, it is interesting to know where the programmer gazes the most or where he moves his eyes. Eye trackers track eye movements and provide a pattern of tracks that are analysed by researchers [27].

Based on the above-mentioned monitoring techniques, it is concluded that the utilization of machine learning (ML) and deep learning (DL) methodologies is prevalent in the analysis of collected datasets, enabling the generation of predictions based on these datasets. Therefore, the primary task of the supervised machine-learning function is to predict or classify the labels associated with any given input data [28]. Machine learning models can be used to classify the level and experience of programmers, whereas deep learning techniques can be used to extract features from fMRI and EEG data.

3. Research method

To answer these research questions, a systematic literature review is conducted in various phases. The inspiration for SLR is taken from a study [29]. Initially, the scope of the study is decided along with the research questions that need to be answered. Based on the initial process, a literature search is conducted to identify studies based on various search strings related to the scope of work. Irrelevant documents were then removed by screening keywords, abstracts, and titles. Subsequently, the remaining relevant publications were analysed thoroughly by full-text reading, and irrelevant studies were removed. Finally, answers to the research questions are provided after extracting relevant information from the publications.

3.1. Search strategy and data sources

We began by identifying useful search phrases to generate a corpus of relevant research analysing the cognitive skills and emotions of programmers while comprehending the source code. The search phrases have been divided into three different sets. The first set of search keywords contains "code comprehension" or "code understandability, as the main focus of our study is to understand or interpret the working of source codes. The second set of keywords includes "cognitive skills," "emotions" or "psychology" as these mental processes affect code comprehension. The third set has keywords like "developers" or "programmers" as the programmers use various mental processes to comprehend the software code. Similarly, to remove bugs and loopholes, the codes are inspected at various levels of the SDLC. Hence, the third set also contains "software maintenance" or "software development". All of these sets are conjugated with each other using the "AND" operator while the keywords within a set use the "OR" operator to depict the variations in search strings. The search criteria are shown in Fig. 1.

The search is conducted across five databases: ACM, IEEE, Springer, Google Scholar, and Scopus. The search query is used to search for relevant articles by applying it to the title, abstract, and keywords. The search process included several aspects of the search engines, such as pluralization and variants. The search results and the inclusion and exclusion of publications are presented in Fig. 2.

3.2. Inclusion-exclusion criteria

Certain criteria have been employed in this study to ensure that only relevant papers are included. A criterion to include or exclude publications, as shown in Fig. 2, is presented in Tables 1 and 2, respectively. Backward snowballing has also been used in later stages [30] to add more relevant studies. The references of relevant publications have been inspected to obtain suitable articles related to the scope of our study [31]. Papers that discuss code comprehension, emotions, cognitive psychology, and BDI are taken into account. To address the research questions, we conducted a five-stage systematic literature study.

1. We began by settling on the scope and questions that would guide our investigation.
2. In the second stage of our study, we conducted a systematic literature review to identify the first corpus based on this scope.
3. We filtered the original corpus by removing duplicates and screening the articles using keywords, abstracts, and titles. Duplicates with similar abstracts have been rejected, and very few studies with extended work are considered. To improve the quality of the corpus, backward snowballing is used to obtain more work-related studies.
4. Next, we performed a comprehensive systematic literature review, utilizing categorization systems derived from our research questions and discarding any remaining papers that did not contribute to answering those issues.
5. Finally, we extracted the information required from the corpus's scholarly output to address our research questions.

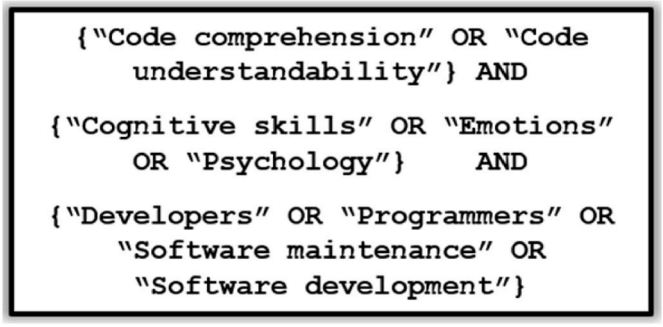

Fig. 1. Search strings.

4. Related works

Recently, researchers have shown an interest in the intriguing issue of code comprehension. The current body of research is small but encouraging. Scientists have utilized multiple approaches to decipher the enigma of the human mind behind deciphering a code, and their conclusions have been conclusive. To answer the research questions defined in our study, a systematic literature review is conducted in the following subsections.

The purpose of this section is to ascertain the impact of emotions, cognitive psychology, and BDI on the code comprehension tasks performed by programmers. In this context, the research studies pertaining to these topics have been examined to identify and analyse the overall and significant correlations among them. The research studies presented in this section has further been divided into four sub-sections. In section 4.1, studies related to code comprehension using various monitoring techniques have been presented. Section 4.2 discusses about the research work done by the researchers on emotions in general or in relation to code comprehension and other domains. Section 4.3 is about cognitive skills in the domain of psychology and how these skills should be measured. This section also includes few studies which explore the relationship of cognitive psychology and code comprehension. Section 4.4 discusses the studies related to BDI features.

4.1. Review on studies related to code comprehension

Understanding the whole process of code comprehension is not simple, as there are many aspects upon which the entire enigma builds. Researchers must have a deep understanding of all related concepts, which is why a systematic literature review is important. In the research work done so far by different researchers, it is seen that they have tried various approaches and worked on various points.

In [32], code comprehension is performed by detecting vulnerabilities in the code, and the detection of vulnerabilities is performed on the existing datasets, which concludes that the quality of the dataset matters significantly while performing code comprehension. A neural network model was used to train and obtain results from these datasets because machine learning models are prone to bias in data. The authors of [23] explored more code comprehension techniques and performed code summarization, search, and statement separation. The fold2vec model detects code misconduct by using deep learning. In Ref. [33], the authors used a block model to hone students' code comprehension skills. The block model mentioned in Ref. [34] is an instructional framework for describing how readers of computer code develop an understanding of the code. The block model is directly related to programming knowledge and schema evolution. The atoms of a language are still learned by novice programmers. Schemas are developed by programmers to help them reason the patterns they discover in the blocks and the relationships between them (concrete operational stage). Finally, at the macro level (the formal operational stage), students gain an understanding of the interconnectedness of the various components of a program. The study concludes that the block model helps students hone their overall code comprehension skills. Researchers in Ref. [32] focused on the quality of the dataset; similarly, the authors in Ref. [35] focused on a to-the-point dataset for code comprehension. Students' ability was measured by their reading skills and the correctness of their responses. One study [36] investigated three case studies and found that using the conditions in place of loops requires less effort during comprehension. If nested loops are used and eye movements are monitored, then these loops become the main area of focus for programmers. It also concludes that changing the variable names of functions does not affect the performance. Machine learning has been used to classify data [28]. This article lists the merits of using wearable and lightweight

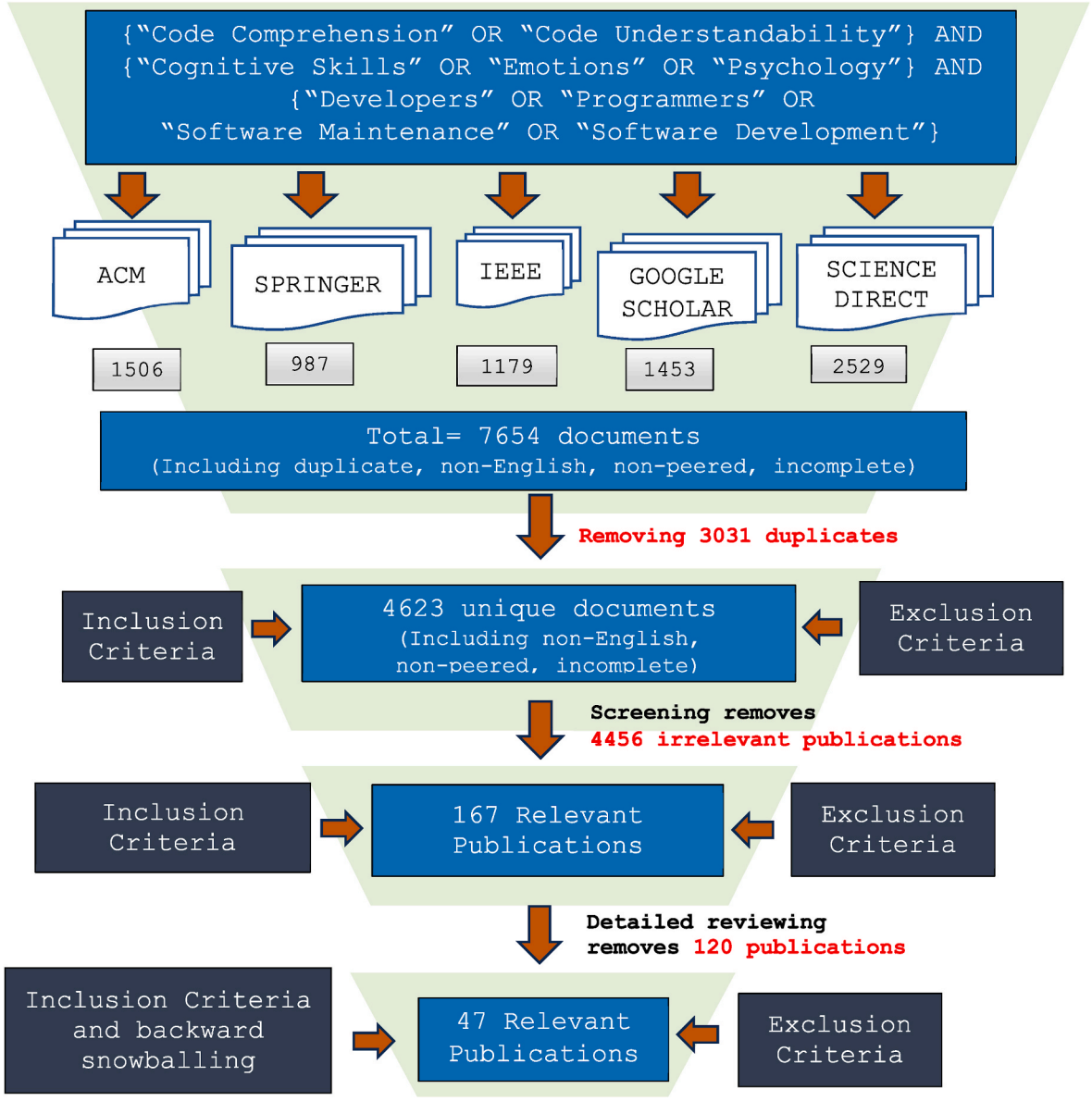

Fig. 2. Inclusion-exclusion criteria.

Table 1
Inclusion criteria.

NO.	Inclusion CRITERIA
1	Papers published between 2003 and 2023
2	Papers that analyse cognitive skills while comprehending source codes
3	Papers that observe the emotions and their impact on code comprehension
4	Papers which are conference proceedings and scientific journals as potential studies

Table 2
Exclusion criteria.

NO.	Exclusion CRITERIA
1	Papers not written in the English language
2	Papers that are not the primary studies related to work scope
3	Incomplete papers, in press papers and non-papers
4	Papers not included in our university subscription
5	Duplicate articles

sensors while comprehending coding tasks. Intelligent computing methods are not restricted to classifying programmers; however, as mentioned in Ref. [37], they can also be used to detect code smells present in the code. Code smells are not bugs technically; they are referred to as weak pieces of code that can cause vulnerabilities in software. For lengthy codes, it has now become easier to use deep learning techniques to quickly and efficiently identify code smells. These methods can also be used to compare the results obtained with heuristic approaches, as stated in Ref. [38]. Instead of lengthy code snippets, researchers in Ref. [39] tokenized code snippets using neural networks. Although the neural network can understand the generated parse tree, major improvements are still sought to learn code syntaxes more precisely. A study [40] mentioned that smelly and spaghetti codes are dangerous for software. These are weakly structured codes that are typically written by inexperienced programmers. The study concludes that the presence of spaghetti code can reduce the efficiency of programmers. Code manipulation, code reading, response time, and error rate are some of the various comprehensive tasks and metrics used in Refs. [24,41]. A previous study [42] conducted some independent studies and corroborated that programmers use code cues while

debugging and that the cues are task-dependent. The study also concluded that the integrated development environment does not have a significant impact on developer performance. The purpose of these studies was to identify the core problems related to code comprehension while maintaining the software. The researchers of [28] mentioned the use of lightweight sensors. To update this list of sensors, many researchers have used eye-tracking sensors to obtain results for cognitive skills while comprehending various code snippets. Study [43] used recursive and iterative code snippets to learn the behavioural patterns of eyes and their tracking movements. The study finds that the bottom-up approach is a prominent skill used by programmers who are experts in coding, while novice programmers use linear code reading or a top-down approach. These results corroborate the claims made in Ref. [27]. A study in Ref. [44] used eye tracking and found that comparing programs obscured by atoms to functionally identical versions reduced the time spent on the atom-containing section by 38.6%. It took 28% fewer attempts to obtain the correct answer. However, the obscured iteration was more difficult to decipher and authenticate, as shown by the more horizontal regressions in the atomic area. The authors in Refs. [45,46] used gaze-based observations to present their novel work, where the primary aim was to explicate the execution and instruction of an algorithmic solution for generating trajectories that depict gaze behaviour. Furthermore, these studies demonstrated the potential of the algorithm to augment the efficacy of an automated program comprehension task. Apart from tracking eye movement, the authors in Ref. [47] studied the blink rate and pupil dilation of participants and found a linear relationship between cognitive load and pupil dilation at constant brightness.

Code comprehension incorporates the use of cognitive skills. Hence, it is fascinating to understand how the brain functions during these tasks. To obtain refined results, researchers in Ref. [26] used a functional magnetic resonance imaging (fMRI) dataset to understand the pattern of brain activation when code comprehension is performed. The authors in Refs. [20,22] also used this technique to understand the workings of the brain. Studies have examined the activation of neurons when various types of code snippets and coding tasks are assigned to programmers. Another brain examination technique often used by researchers is electroencephalography (EEG). EEG electrodes were connected to various sensitive spots on the scalp to obtain data. The electrical signals generated by these electrodes create spikes at the output screen when coding tasks involving cognitive skills are performed, and the neurons of that part of the brain are activated during this process. Researchers in Ref. [48] have tried to study these patterns because various frequency-related changes are observed when code comprehension is performed. The work done by the authors in Ref. [49] concludes that expert programmers have great short-memory skills, and when the programmers are experts, there are fewer activated brain activation areas as compared to novice programmers, as mentioned in Ref. [50]. Studies [25,51] combined eye tracking and EEG data to conduct code comprehension experiments and attempted to find common ground between the datasets of these sensors. Researchers in Ref. [52] also combined fMRI and eye-tracking data but concluded that synchronizing these data and their timestamps requires significant improvement. In summary, collecting data from multiple sensors is very costly, and not every researcher can afford it. In addition, synchronizing EEG, fMRI, and eye-tracking data with each other is a very little-explored domain. The authors in Refs. [53,54] constructed a cognitive attraction network (CAN) model in which weight assignment is performed on various cognitive skills, and based on the performance in these skills, the cognitive load and rank of programmers is predicted. The obtained results were then used to assign projects to programmers based on their rank and cognitive scores.

Hence, it can be concluded that there are several ways to measure programmers' cognitive skills, ranging from simple tasks to complex and long experiments. These experiments yielded various results, and to compute inferences, various computational methods such as ML and DL have become prominent.

4.2. Review on studies related to emotions

Emotions are mental states triggered by neurophysiological alterations linked to different kinds of thinking, sensing, and behaving, as well as to degrees of pleasure and displeasure. The state of mind, or emotional state, is often intertwined with one's disposition, personality, or ability to think creatively. It is necessary to understand how much work has been done in this field. The three-dimensional taxonomy of achievement emotions is described by control value theory (CVT) in Ref. [55]. In this taxonomy, object focus, activation, and valence serve as three axes. Activities or results may serve as the focus of attention. The activity of programmers is involved, and the activity's outcome should be the primary focus when studying emotions. Emotions have positive or unpleasant valence. The term "activation" is used to describe the physiological arousal that one feels when experiencing an emotion. Feelings of joy, optimism, pride, anxiety, anger, and shame are all examples of stimulating emotions, whereas feelings of relief, hopelessness, boredom, and satisfaction are examples of calming emotions.

The central tenet of the theory is that programmers feel good emotions when they are actively engaged in things; they find it meaningful. Similarly, when programmers do not value certain activities or feel out of control, they experience negative emotions. Value appraisals consider how meaningful an activity is to the individual performing the evaluation, whereas control appraisals focus on how much control the individual has over the action and its results. Emotions are influenced by various control and value appraisals. Appraisals, emotions, and results are the three main parts of the CVT. Emotions are influenced either directly or indirectly by appraisals. Achievement is influenced by emotions, which in turn influence emotions. Later, emotions and appraisals were found to be influenced by success. Since achievement emotions, appraisals, and outcomes are interconnected, there is a reciprocal causal link between them.

A study [15] examined programmers' emotions during code comprehension using the control value theory concept. The study concluded that novice programmers use a trial-and-error strategy to debug codes without knowing the cause of the error. This study corroborates the claim that negative emotions affect the learning process of programmers, and programmers feel demoralized and ashamed upon failing the task. The authors of [56] created a multimodal database model to research emotions using EEG, voice, photoplethysmography, and facial photographs. Four baseline algorithms, probabilistic linear discriminant analysis (PLDA), temporal convolution network (TCN), extreme learning machine (ELM), and multilayer perceptron (MLP), are used to test the performance. The EEG signals captured emotions better. Game-based learning [57] examined emotions and cognition. Compared to non-game tasks, the neurofunctional patterns implicated in emotional processes show considerable activation. In Ref. [58], Ortony, Clore, and Collins (OCC), Sherer, and Roseman created a weighted model for computing five primary emotions: happiness, anger, sadness, fear, and surprise. The international survey on emotions antecedents and reactions (ISEAR) and real-world data confirm this study. Games allow us to observe emotions quickly in different situations. Games demand cognitive skills for reaction times. The authors in Ref. [59] found that declarative memory influences core affect, and emotional decisions occur faster than conscious processing. In Ref. [60], uncivil comments received online provoke hostile cognition, which affects behaviour and emotions; however, when made by the individual, no hostile emotion trigger was seen. It is intriguing whether someone makes emotional decisions purposefully or automatically.

4.3. Review on studies related to cognitive psychology

The field of cognitive psychology investigates how the brain functions in relation to cognition, such as education and communication.

Different people have different learning styles, memory capacities, and communication modes. Cognitive psychology studies how people act based on their perceptions and interpretations of information. Psychologists, such as the ancient philosophers Aristotle and Plato, are interested in the acquisition of knowledge, although their methods are significantly more sophisticated.

Psychology and cognitive skills are deeply connected. A study [61] explored the connection between cognitive skills and cognitive load of programmers. Cognitive load refers to the number of resources that working memory needs to perform a particular task or a number of tasks. When code comprehension is completed, it uses various cognitive skills that act as resources; hence, the more cognitive skills we use, the greater the cognitive load will be. The researchers in Refs. [61,62] used eye-tracking methods to measure the cognitive load of programmers. The studies concluded that user fixations, saccades, and pupil reactions are important features for identifying challenging parts of the code. The work done by the authors in Ref. [63] emphasizes task descriptions and interactive decision assistance. A previous study [64] created declarative cognitive architecture. The model includes the storage, retrieval, and encoding steps. Working memory, or short-term memory (STM), is the mental capacity to maintain a set amount of information in the head at any given time. This helps with learning, reasoning, and getting ready to take action. Mid-term memory (MTM) allows faster data manipulation, including storage, retrieval, and updating. Because it can only store a fixed but larger amount of information than STM, the amount of information can process data very quickly. Data are transferred from short-term to long-term memory and stored indefinitely in persistent memory (PM) via a persistence signal triggered periodically by a circadian rhythm process. This level does not have a hard storage cap as the last one did, so it takes longer to manipulate the data.

The task set reduces the response time, and the STM retrieves faster than the MTM and PM. One study [65] examined the teaching area to understand collegiate cognitive psychology. The two types of teaching approaches are textbooks and micro-courses. The major goal was to observe the cognitive impact using both methods because it influences time, accuracy, and efficiency. Cognitive psychology harnesses the information stored in the brain. The psychology of programmers is imitated in Ref. [66] as students are given the role of programmers. Agile software records cognition style and interaction patterns. The methods discussed in Ref. [67] examine the predispositions produced by an automatic cognitive system versus a deliberate technology assessment system. The study observed the behavioural intentions and effort expectancy of the participants.

4.4. Review on studies related to belief-desire-intention

The belief-desire-intention (BDI) model comprises of several mental qualities and relationships. Bratman proposed the BDI model, a philosophical framework for practical reasoning. Beliefs, desires, and intentions comprise the mental attitude component of the BDI paradigm. Beliefs are knowledge or information about the world. Intentions and actions are processed sequentially, and the environment is updated. To understand the work done in this area, a literature survey is conducted.

Understanding the BDI is crucial for understanding the source code. The BDI transformer agent (BTA) model in Ref. [68] uses the BDI to rank models and select the best. The genetic algorithm improves model correctness. Human behaviour affects beliefs, wants, and intentions. In Ref. [69], intelligent agents that support these are upgraded to present an ABC-EBDI paradigm. It incorporates the psychotherapy paradigm, as mentioned in Ref. [70], ABC model, and other affective theories. The basic idea behind the ABC model in Ref. [71] is that external events (A) do not cause emotions (C) but beliefs (B) and, in particular, irrational beliefs (IB). This framework models human conduct in terms of acts and expressions. In a typical model of cognition, one study [72] focused on the metacognition model. In order to select the most suitable project [73], the Selection of Maintenance Engineer (SME) model predicts the attitude of the maintenance engineer based on the selection index for the given set of data in terms of reliability, reputation, and BDI. Artificial neural networks (ANN) and clustering techniques were used to perform the computational part of this study. An (ANN) is a computational model based on biological neural networks in the brain that performs tasks such as pattern recognition, classification, regression, and decision-making. The ANN has input, hidden, and output layers of interconnected points (i.e., artificial neurons). It uses learning algorithms to independently adjust and learn from new inputs, which makes it suitable for nonlinear statistical data modelling [74]. Similar to previous work, a study [75] used BDI to rank online services. It also introduces a novel framework for reputation as a measurement parameter. The above literature survey shows that there is no standardized dataset for these investigations.

4.5. Key findings of SLR

After conducting a systematic literature review, it can be concluded that cognitive skills are very important for understanding how a programmer comprehends a source code. Every cognitive skill tells a new angle on how programmers proceed with the information they have and how they are going to solve the tasks. Programmers use various approaches according to their knowledge and experience levels to comprehend the code snippet. An expert programmer often uses a bottom-up approach, whereas a novice programmer attempts to implement a top-down approach [27]. These approaches activate different parts of the human brain, and various cognitive skills of programmers are utilized. Thus, programmers' approach matters when comprehending code and learning about cognitive skills.

Researchers must have a well-formed structured dataset and a good understanding of the experimental environment. The experimental setup should cover all possible cognitive skills to solve the proposed problem. Code comprehension involves a variety of tasks, such as code reading, writing, debugging, and understanding, which use different cognitive skills. For example, when collecting data using an EEG sensor, a sensor with more channels will provide more detailed and synchronized data [25]. In recent years, experimental setups like fMRI, EEG, and eye tracking have become very prominent among researchers for code comprehension. These setups can collect data in various forms, which cannot be collected through coding assignments. Researchers [26] are now able to look inside a programmer's brain and see which part of the brain is activated when some cognitive tasks are performed. Researchers are able to track and observe brain frequencies, eye movements, and pupil dilations, and have tried to collect valuable data by synchronizing the data collected from these sensors. The processing of these data, including testing and training, is carried out with only a limited amount of participation of participants in the experimental setups using intelligent computing methods such as machine learning and deep learning. Therefore, more participation is required to obtain more generalized results through various monitoring techniques.

Lastly, it has been discovered that there is a dearth of research investigating the correlation between code comprehension, emotions, cognitive psychology, and BDI. In view of this, our focus in this work is to capture the notion of how emotions can be related to code comprehension and cognitive skills. The rationale behind the notion of studying emotions in context with code comprehension stems from two major reasons:

1. Emotions can significantly influence a programmer's engagement, motivation, and overall experience with coding. Studies have shown that programmers' performance is significantly influenced by their emotions; positive emotions can enhance creativity and problem-solving abilities, while negative emotions can lead to reduced cognitive skills and reliance on hit and trial methods, thereby hindering the comprehension. Therefore, there is a significant necessity

to investigate how emotions affect the comprehension skills of programmers.
2. Capturing the essence of a programmer's emotional state is comparatively more straightforward than discerning their cognitive psychology and belief-desire-intentions. From programmers' perspective, articulating their emotional state is simple and direct, unlike describing the intricate cognitive processes involved in deriving logic during code comprehension tasks.

5. Key findings and insights from research questions

This section highlights the key findings from the conducted SLR for the respective research questions defined in section 1. The findings for the respective research questions are discussed in the sub-sections.

5.1. RQ 1: What monitoring methods have been used for code comprehension?

The monitoring methods used for code comprehension are shown in Fig. 3. There is a clear preference for evaluating programmers' analytical skills through coding exercises (29%). To perform code comprehension, programmers need to have good knowledge of the programming language used in the code snippet, and the programmer should have good cognitive skills. Skills such as bottom-up and top-down approaches are used by programmers to understand the source code in the best possible way; hence, they define their experience level [76]. Experienced programmers tend to read the code snippet function calls to generate an idea regarding the code snippet, but on the other hand, a novice programmer likes to read a code snippet from the beginning and prefers to go line by line. In addition, code reading and comprehension skills depend on the keywords used as functions in the code snippets of study [20], as using familiar keywords results in less brain activity and programmers are able to comprehend easily. Code complexity also plays a significant role in testing programmers' cognitive skills, such as problem-solving, reasoning, recalling, and analytical power. An experienced programmer can understand code complexity using techniques, concepts, tricks, etc., and combine these skills with a bottom-up approach to simplify things in a very short time. On the other hand, a novice programmer might complex things using the top-down approach with limited knowledge of concepts, and will take more time to solve the coding problem.

Only twenty-one percent of the researchers have developed their own models to analyse the human brain. Other common approaches to gauging mental acuity include electroencephalography (8%), functional magnetic resonance imaging (10%), eye-tracking (16%), and a combination of modalities (6%). Methods such as electroencephalography (EEG), eye-tracking, and functional magnetic resonance imaging (fMRI) are a part of several techniques used in a single experiment.

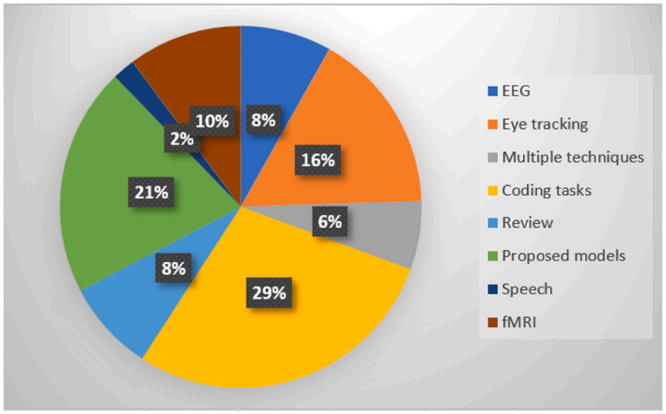

Fig. 3. Monitoring techniques.

Based on the studies that have been selected for the research work, it can be concluded that there is no visible trend regarding the use of a certain type of monitoring technique as of now by the researchers. After 2015, researchers began using techniques such as EEG, fMRI, eye tracking, or a combination of these techniques more frequently. Fig. 4 corroborates that researchers now prefer sensorial data along with the comprehension tasks designed by them.

5.2. RQ 2: Which performance metrics have been used in the research works?

Performance metrics are examined to gauge the effectiveness of the work. The effectiveness of the findings is indicated by performance metrics. The statistical tests, confusion matrix, time measure, correctness of response, complexity metrics, brain activation, eye gaze and saccades, experience, and emotions are just a few of the metrics employed, as shown in Fig. 5. Various performance metrics investigate calibration, productivity, and various technical, psychological, and emotional parameters. In a simple language, the correctness of the response calculates the number of correct answers given by a programmer while comprehending a task. In a more advanced interpretation, the coding task contains various types of code snippets. This may include snippets related to code reading, writing, and debugging. Therefore, the correctness of the response includes the answers from all subsets of code comprehension and makes it a versatile performance metric for investigation. To further enhance the results, programmers are asked to perform the given coding tasks within a time constraint [44]. The time measure acts like a pressure situation that can cause activation of multiple brain areas and various cognitive skills. Therefore, the time measure had a significant impact on the correctness of the response.

Assessing code comprehension skills and psychological aptitudes necessitates an understanding of programmers' cognitive skills. Enhanced cognitive skills are perceived to correlate positively with the expertise of programmers within their respective domains. To determine the expertise level of programmers, programmers are classified based on the results collected from various coding and cognitive tasks. One of the main goals of classification is to compute the cognitive load of programmers such that suitable projects can be assigned to them based on their skill level. This classification is performed with the help of machine-learning methods, as they are capable of performing computations for larger datasets in a very time-efficient manner. The ML results are analysed by generating a confusion matrix. The confusion matrix contains performance measures such as accuracy, f1 score, precision, and recall. Hence, the confusion matrix is a reliable performance metric and has been used by various researchers in their works.

Proposing and testing the hypothesis or statistical testing are prominent and efficient methods to validate the results of this study. Hypothesis testing includes various parametric and non-parametric tests such as the *t*-test, paired *t*-test, Wilcoxon test, Pearson correlation, and Mann-Whitney test. To validate the test results, the value of significance (p-value) is defined as a threshold value. The acceptance and rejection of a hypothesis then depends on the results obtained with respect to the p-value. For example, a research hypothesis is defined in the initial stages of research, in which keywords play an important role in code comprehension. After collecting data and obtaining results from parametric or non-parametric tests, they are compared with the p-value. If the value is greater than the defined p-value, the hypothesis is accepted that keywords are important for code comprehension; otherwise, the hypothesis is rejected.

Emotions and brain activations are considered performance metrics because when a code comprehension task is performed, programmers use various cognitive skills that cause the brain to activate neurons in those areas. The human brain is divided into areas known as the Brodmann area (BA), and researchers use fMRI, EEG, and various other scans to identify the related cognitive areas that perform highly during code

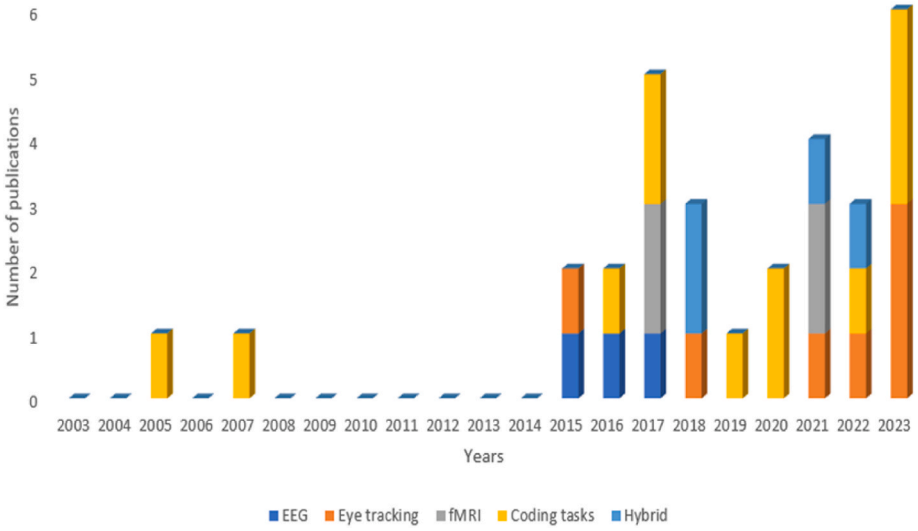

Fig. 4. Trends of using monitoring techniques.

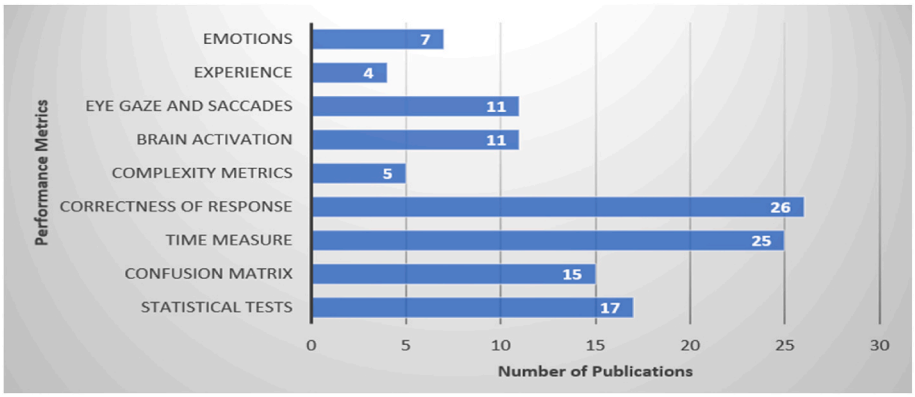

Fig. 5. Performance metrics.

comprehension tasks. On the other hand, programmers go through various emotional phases while performing code comprehension tasks. Researchers have studied the relationship between emotions and programmer performance. Programmers' engagement, motivation, and overall coding experience can be greatly influenced by emotions. Studies have demonstrated that the performance of programmers is greatly impacted by their emotions. Positive emotions have the potential to improve creativity and problem-solving skills, whereas negative emotions can diminish cognitive abilities and result in a dependence on hit and trial methods, thereby impeding comprehension.

5.3. RQ 3: What are the various computational methods used in various works?

Researchers have conducted experiments to draw meaningful conclusions. To offer meaningful research, data gathered from various approaches must be processed using certain computational models. Various computational methods have been used in the reviewed articles, as shown in Fig. 6.

Deep learning is the most commonly used computational model used by the researchers in this study. It has been used in 43% of the research articles in this study. Another well-known intelligent computational technique is machine learning, which is used in 30% of the research works. The data collected through various sensors, such as fMRI, eye tracking, and EEG, contain considerable noise. Whenever these sensors collect data, they select actual data and additional data that are

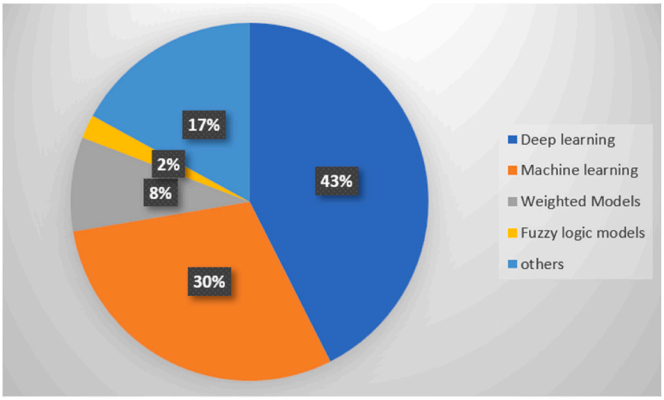

Fig. 6. Various Computational methods.

irrelevant to the study. Comprehension tasks using multiple sensors require data that are synchronized at the same time stamp. The remaining data is mostly noise (unwanted data). Therefore, it is necessary to obtain accurate data from that heap. This is where intelligent computing methods have come into play. Deep learning can directly extract data from raw signals and provide noiseless data by extracting the data through hidden layers. Similarly, when fMRI data are collected by researchers, the data are of 3D or 4D volumes. Therefore, DL is used

to extract the relevant features from the collected images without pre-training. Machine learning models are also used to extract features related to sensorial data. In addition to the noise in the data, there are other problems. The collected data do not have all attributes or possess the same range of values. One feature may range between a smaller scale, whereas some other features may show a completely different range of values. Machine-learning methods have been used to solve this problem. Machine learning and deep learning methods are used to normalize the data. Normalization refers to bringing every attribute value to be used within a common range. Usually, the normalized data lie between zero and one. Machine learning is also used to extract features related to spectral power, time-frequency distribution, and data synchronization. This process of extracting features, normalizing, cleaning data, and removing noise is known as data pre-processing. These methods are used to classify the results based on the dataset. The models detect whether there is a hidden pattern in the data. They detected patterns and classified the data accordingly. These models have been used to classify programmers based on experience or to determine which code comprehension method between the bottom-up and top-down approaches is more prominent. One of the most important features of these methods is that they can process a large amount of data in a time-efficient manner. Eye tracking is another sensorial data that can be processed in a similar way as data related to fMRI and EEG are handled by ML and DL. Seventeen percentage of researchers have defined their own models based on existing theories and models and have concluded various results related to the study [58] domains. Weighted models (8%) are mathematical models in which the results are calculated by assigning weights to attributes related to the study, as shown in the case study. Mathematical models are used to compute the cognitive loads of programmers such that they can be assigned projects based on their cognitive and programming skills. To understand the relationship between emotions and cognitive skills while comprehending the coding snippets, a case study is presented in Section 6.

6. Methodology and implementation: a case study

Studies [15,16] conducted experiments to measure the programming and cognitive skills of programmers and the relationship between emotions and those skills while comprehending code snippets. Taking inspiration from that study, an experiment is conducted to find the relationship between the higher-order cognitive skills of participants and their emotions while comprehending a coding assignment. The participants used their reasoning, problem-solving, and critical thinking skills to solve the given tasks. An architectural model is defined in Fig. 7, which aims to measure the cognitive load. The comprehensive and emotional reflective indices are calculated based on the collected dataset. After combining the obtained indices, a cognitive index is obtained, which is used to categorize participants according to their level of expertise. Various machine learning models have also been employed to verify empirically obtained results.

The data to conduct this experiment has been collected using a Google form and circulated to the students of the computer science and engineering department studying in the third year, research scholars, and professionals working in various IT industries. After reviewing the submitted forms, data from 40 students, seven research scholars, and three working professionals is compiled. The Google form contained ten basic but tricky-coding questions related to C programming. These included questions related to basic coding, arrays, conditional statements, logical statements, and nested loops. Participants are asked to provide the output of the questions. Along with the coding assignment, some questions related to emotions have also been added so that the participants could describe or express their emotions related to the assignment. Participants are asked to express their emotions before, during, and after the test. The performance of the participants is then checked. Based on the collected data, two types of relative indices, comprehensive and emotional, are calculated. The comprehensive

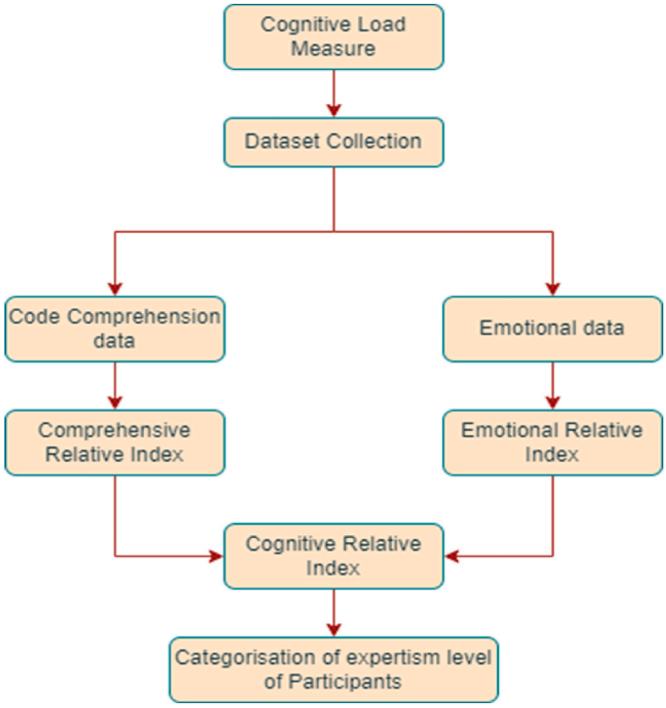

Fig. 7. Architecture of case study.

relative index (R1) includes the number of correct answers given by the participants. The R1 values ranged between 0 and 1. The expression for calculating R1 is shown in Equation (1).

$$R1 = \frac{Correct\ answers}{Total\ answers} \tag{1}$$

To calculate the value of the emotional relative index (R2), as shown in Equation (2), the emotions have been categorized into three subclasses: positive, negative, and neutral. Weights are assigned to these emotional subclasses, and similarly, weights have been assigned to the emotions before, during, and after the experiment. Emotions before (w1) and after (w3) have been given a value of 0.25, whereas the value of emotions during the test (w2) is chosen as 0.5, as the impact of emotions during the comprehension of assignments is greater, as shown in Table 3.

$$R2 = (sub\ class \times w1) + (sub\ class \times w2) + (sub\ class \times w3) \tag{2}$$

For example, one participant felt positive before the test and became nervous during and after the test. Then, the calculation of R2 is performed using Equation (2).

$$R2 = (0.6 \times 0.25) + (0.3 \times 0.5) + (0.3 \times 0.25)$$

Once the values of R1 and R2 are calculated, the values of cognitive relative index (R3) are obtained. R3 is calculated using the values of R1 and R2 with a selected value of $\alpha = 0.4$, and the range of α is between 0 and 1 (i.e., $0 < \alpha < 1$). Therefore, the expression for R3 is given by Equation (3).

Table 3
Weight assignment.

Weight assignment	
Positive emotions	0.60
Negative emotions	0.30
Neutral emotions	0.10
Emotions before assignment	0.25
Emotions during assignment	0.50
Emotions after assignment	0.25

$$R3 = (\alpha \times R1) + ((1 - \alpha) \times R2) \quad (3)$$

The calculated value of R3 lies between 0 and 1; therefore, the value of R3 is used to categorize the participants according to their expertise level. The categorization of expertise is done in five classes, as shown in Table 4, and a sample of data results is shown in Table 5.

The results showed that 68% of the participants had an average expertise level, 24% achieved a low level of expertise, and 8% had high knowledge of the C programming language. In this experiment, none of the participants had a very low or very high level of expertise. The conclusions are shown in the pie chart in Fig. 8.

To support our claim, various machine learning and ensemble models have been applied to the collected dataset. These computational techniques are applied to the independent features of the dataset; therefore, features are selected to categorize the data into five different classes of expertise. Ensemble learning models such as bagging, AdaBoost, and voting classifiers have also been used to obtain results. Table 6 presents the results. As per the conclusions of the case study, no participant achieved the expertism level of the very high and very low categories. The results in Table 6 present the values of the confusion matrix parameters for the remaining three classes, that is, low, average, and high categories.

Therefore, this study concludes that programmers' emotions play a crucial role in code comprehension tasks. If programmers show positive emotions throughout their experimental work and have achieved good comprehension value (R1), then their cognitive relative index value (R3) categorizes them at a higher level of expertise. On the other hand, if the programmers show a variety of emotions during the experimental phase, then based on the results obtained from R3, the skillfulness of programmers is categorized.

7. Threats to validity

Threats to validity encompass various factors that could hinder the interpretation of research findings and the establishment of a causal relationship between the independent variable and observed changes in the dependent variable. Our study is subject to threats to internal validity (limitations of search strings, study inclusion-exclusion, and data extraction), external validity (generalizability), conclusion validity, and reliability validity.

Internal validity deals with problems related to search strings, inclusion-exclusion criteria, and data extraction. Quality publications have been selected to counter threats to internal validity. To find relevant publications related to the scope of our study, combinations of various search strings are developed. The search keywords were discussed and agreed upon by the authors and finalized keywords were used to construct valid search strings. The search strings contained metonym searches like "code comprehension" or "code understanding," "software maintenance" or "software development," and "developers" or "programmers" to expand the search relevance in five different databases which include ACM, IEEE, Scopus, Google Scholar, and Springer. For the selection of publications through inclusion-exclusion criteria, a systematic procedure is adopted to remove threats to validity. Through inclusion-exclusion criteria, duplicate publications were removed first, and then the screening process is conducted to find the relevant publications related to our work. To select more potentially relevant publications, the authors read the important sections of the publications and chose the studies that were needed for our research questions and scope of study based on the quality assessment factors designed by the authors. Another threat to validity is the cost of monitoring code comprehension techniques. Apart from coding assignment tasks, other techniques such as EEG, fMRI, and eye tracking are very expensive and complex.

External validity corresponds to the generalizability of our experimental results, as concluded in section 6. In our research, we collected cognitive and emotional data using an online form. The form contained code snippets for participants to comprehend. The collected database is then used in an experimental case study that measured the cognitive load of programmers by testing their higher-order cognitive skills and the relationship of emotions with those cognitive skills while comprehending the source code. The experiment is limited to a few participants and did not cover all the aspects of cognitive skills. Smaller code snippets were also used to obtain the results. Hence, the generalization of longer code snippets may differ in the results.

Conclusion validity refers to making incorrect conclusions and ensuring the replicability of the study. For the former, we covered a variety of potential problems that could result in incorrect conclusions in the context of threats to the internal and external validity. Section 6 provides a comprehensive description of the research procedure used to reproduce this mapping study in its entirety.

Reliability validity refers to the extent to which the same data yields consistent results when replicated. This study is an attempt to measure the cognitive load of programmers when emotions influence cognitive skills. The emotions of programmers vary over time, which, according to our research, may yield different results when repeated.

Table 4
Categorization of programming expertism.

Expertism level	Value of R3
Very low (VL)	0.00–0.20
Low (L)	0.21–0.40
Average (A)	0.41–0.60
High (H)	0.61–0.80
Very high (VH)	0.81–1.00

Table 5
Sample data results.

Participant	Comprehension Relative Index (R1)	Emotional Relative Index (R2)	Cognitive Relative Index (R3)	Categorization of participants
1	0.6	0.38	0.47	Average
2	0.8	0.35	0.53	Average
3	0.8	0.38	0.55	Average
4	0.9	0.38	0.59	Average
5	1.0	0.60	0.76	High

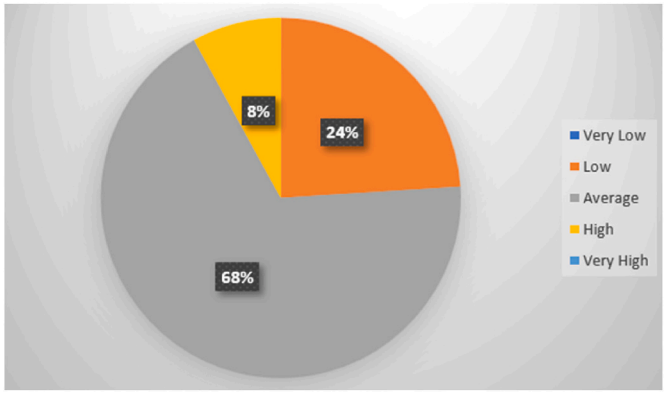

Fig. 8. Categorization of participants.

8. Conclusion

This study presents the findings of an empirical study that focuses on determining the expertise of a programmer based on its ability to comprehend the code under the influence of various emotional states. The study categorizes programmers by implementing machine learning and ensemble learning models based on their coding and cognitive skills by computing their cognitive load. To better comprehend this idea, a brief SLR of existing techniques to monitor and measure cognitive skills

Table 6
ML techniques and confusion matrix parameters.

ML Models	Accuracy	Precision			Recall			F1- Score		
		Low	Average	High	Low	Average	High	Low	Average	High
SVM	0.87	1.00	0.82	1.00	0.60	1.00	1.00	0.75	0.90	1.00
Decision Tree	0.80	0.67	0.88	1.00	0.80	0.78	1.00	0.73	0.82	1.00
KNN	0.80	1.00	0.75	1.00	0.40	1.00	1.00	0.57	0.86	1.00
Logistic Regression	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Random Forest	0.87	1.00	0.82	1.00	0.60	1.00	1.00	0.75	0.90	1.00
Bagging	0.93	1.00	0.90	1.00	0.80	1.00	1.00	0.89	0.95	1.00
Adaboost	0.80	0.75	0.80	1.00	0.60	0.89	1.00	0.67	0.84	1.00
Voting	0.93	0.83	1.00	1.00	1.00	0.89	1.00	0.91	0.94	1.00

is presented in this paper. The reviewed publications cover cognitive skills, such as emotion, psychology, and BDI, as well as code comprehension. From the empirical study and SLR conducted, it can be concluded that performance metrics such as time measure and correctness of response (accuracy) tend to play a crucial role in determining code comprehension skills. The actual ability of programmers is tested when they are able to solve the bugs in a limited time frame, and the issues are resolved without altering the actual outline of the product. Computational methods, such as deep learning and machine learning, are more prominent methods attributing to intelligent computing methodologies. The essential constraints in code comprehension, such as the number of participants, the size of the code, and the freedom of individuals to move around, are the most prevalent difficulties encountered in such experimental setups. Methods such as functional magnetic resonance imaging, electroencephalography, and eye tracking have emerged as primary tools for academic research. Further, from the empirical study, it can be concluded that programmers' emotions affect their performance. Programmers showing more positive emotions have better comprehension skills and thus are categorized relatively in a better class than those who possess negative emotions while comprehending the coding tasks.

Lastly, a few observations are listed below, which can pave the way for future work on code comprehension and cognitive skills.

- Finding efficient methods, such as EEG, eye-tracking, and fMRI, to analyse brain activity with a sufficient sample size can be the focus of future research.
- Code complexity, code tracing, code completion, and speech recognition are among the few software and performance metrics that can be used as criteria for evaluating a programmer's cognitive skills.
- Smaller code snippets have been used in the case study. Thus, it would be interesting to work on longer code snippets along with deep learning techniques, such as large language models (LLMs) in the near future. Large language models (LLMs) refer to a category of artificial intelligence (AI) algorithms that employ deep learning methodologies and vast datasets to comprehend, summarize, produce, and predict novel content. With the help of LLMs, a summary of the source code can be generated for programmers to understand the purpose of the code without much detail. LLMs use large corpora of knowledge to recognize patterns and make predictions to provide suggestions for code completion. Hence, LLMs can play an important role in software maintenance by identifying bugs and weakly written codes and generating solutions to debug the code using its predictive model. By providing a summary and documentation of the software code, the programmer can handle the software maintenance role in a time-efficient manner. Thus, LLMs for code comprehension will pave the way for future research.

CRediT authorship contribution statement

Divjot Singh: Writing – review & editing, Writing – original draft, Investigation, Formal analysis. **Ashutosh Mishra:** Supervision, Methodology, Conceptualization. **Ashutosh Aggarwal:** Writing – review & editing, Supervision, Methodology.

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.

Data availability

Data will be made available on request.

References

[1] K.H. Bennett, V.T. Rajlich, Software maintenance and evolution: a roadmap, in: Proceedings of the Conference on the Future of Software Engineering, ICSE 2000, 2000, pp. 73–87, https://doi.org/10.1145/336512.336534.
[2] N.F. Schneidewind, The state of software maintenance, IEEE Trans. Software Eng. SE-13 (3) (1987) 303–310, https://doi.org/10.1109/TSE.1987.233161.
[3] R. Malhotra, A. Chug, Software maintainability: systematic literature review and current trends, Int. J. Software Eng. Knowl. Eng. 26 (8) (2016) 1221–1253, https://doi.org/10.1142/S0218194016500431.
[4] S. Velmourougan, P. Dhavachelvan, R. Baskaran, B. Ravikumar, Software development life cycle model to improve maintainability of software applications, in: Proceedings - 2014 4th International Conference on Advances in Computing and Communications, ICACC 2014, 2014, pp. 270–273, https://doi.org/10.1109/ICACC.2014.71.
[5] P. Ragunath, S. Velmourougan, P. Davachelvan, S. Kayalvizhi, R. Ravimohan, Evolving A new model (SDLC model-2010) for software development life cycle (SDLC), International Journal of Computer Science and Network Security 10 (1) (2010) 112–119.
[6] F. Häser, M. Felderer, R. Breu, Is business domain language support beneficial for creating test case specifications: a controlled experiment, Inf. Software Technol. 79 (2016) 52–62, https://doi.org/10.1016/j.infsof.2016.07.001.
[7] A.N. Johanson, W. Hasselbring, Effectiveness and efficiency of a domain-specific language for high-performance marine ecosystem simulation: a controlled experiment, Empir. Software Eng. 22 (4) (2017) 2206–2236, https://doi.org/10.1007/s10664-016-9483-z.
[8] T. Kosar, M. Mernik, J.C. Carver, Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments, Empir. Software Eng. 17 (3) (2012) 276–304, https://doi.org/10.1007/s10664-011-9172-x.
[9] T. Kosar, S. Gaberc, J.C. Carver, M. Mernik, Program comprehension of domain-specific and general-purpose languages: replication of a family of experiments using integrated development environments, Empir. Software Eng. 23 (5) (2018) 2734–2763, https://doi.org/10.1007/s10664-017-9593-2.
[10] K.B. Gallagher, J.R. Lyle, Using program slicing in software maintenance, IEEE Trans. Software Eng. 17 (8) (1991) 751–761, https://doi.org/10.1109/32.83912.
[11] N. Ragonis, R. Shmallo, The application of higher-order cognitive thinking skills to promote students' understanding of the use of static in object-oriented programming, Inf. Educ. 21 (2) (2022) 331–352, https://doi.org/10.15388/infedu.2022.10.
[12] W. Li, J.Y. Huang, C.Y. Liu, J.C.R. Tseng, S.P. Wang, A study on the relationship between student' learning engagements and higher-order thinking skills in programming learning, Think. Skills Creativ. 49 (July) (2023) 101369, https://doi.org/10.1016/j.tsc.2023.101369.
[13] M. Dahn, D. DeLiema, Dynamics of emotion, problem solving, and identity: portraits of three girl coders, Comput. Sci. Educ. 30 (3) (2020) 362–389, https://doi.org/10.1080/08993408.2020.1805286.
[14] G. Shaw, E. Van Der Poel, Genetic algorithms as a feasible re-planning mechanism for belief-desire-intention agents, in: ACM International Conference Proceeding Series, vol. 28, 30-Sept, 2015, https://doi.org/10.1145/2815782.2815817.
[15] Z. Atiq, M.C. Loui, A qualitative study of emotions experienced by first-year engineering students during programming tasks, J. Educ. Resour. Comput. 22 (3) (2022) 1–26, https://doi.org/10.1145/3507696.

[16] H.G. Lapierre, P. Charland, E.P.M. Léger, Looking 'Under the Hood' of Learning Computer Programming: the Emotional and Cognitive Differences between Novices and Beginners, Computer Science Education, 2023, pp. 1–22, https://doi.org/10.1080/08993408.2023.2214033.
[17] L. Gale and S. Sentance, "Investigating the attitudes and emotions of K-12 students towards debugging,", Proceedings of the 2023 Conference on United Kingdom & Ireland Computing Education Research (pp. 1-7), doi: 10.1145/3610969.3611120.
[18] D. DeLiema, et al., "Debugging as a Context for Fostering Reflection on Critical Thinking and Emotion," Deeper Learning, Dialogic Learning, and Critical Thinking: Research-Based Strategies for the Classroom, 2019, pp. 209–228, https://doi.org/10.4324/9780429323058-13.
[19] M. Nayrolles, A. Hamou-Lhadj, Towards a classification of bugs to facilitate software maintainability tasks, in: Proceedings - International Conference on Software Engineering, 2018, pp. 25–32, https://doi.org/10.1145/3194095.3194101.
[20] J. Siegmund, et al., Measuring neural efficiency of program comprehension, in: Proceedings - 2017 11th Joint Meeting on Foundations of Software Engineering, 2017, pp. 140–150, https://doi.org/10.1145/3106237.3106268.
[21] R. Chmiel, M.C. Loui, Debugging: from novice to expert, SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education) 36 (1) (2004) 17–21, https://doi.org/10.1145/1028174.971310.
[22] B. Floyd, T. Santander, W. Weimer, Decoding the representation of code in the brain: an fMRI study of code review and expertise, in: Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering, ICSE 2017, 2017, pp. 175–186, https://doi.org/10.1109/ICSE.2017.24.
[23] F. Bertolotti, W. Cazzola, Fold2Vec: towards a statement based representation of code for code comprehension, ACM Trans. Software Eng. Methodol. 32 (1) (2023) 1–31.
[24] A. Karahasanović, A.K. Levine, R. Thomas, Comprehension strategies and difficulties in maintaining object-oriented systems: an explorative study, J. Syst. Software 80 (9) (2007) 1541–1559, https://doi.org/10.1016/j.jss.2006.10.041.
[25] Y.T. Lin, Y.Z. Liao, X. Hu, C.C. Wu, EEG activities during program comprehension: an exploration of cognition, IEEE Access 9 (2021) 120407–120421, https://doi.org/10.1109/ACCESS.2021.3107795.
[26] N. Peitek, S. Apel, C. Parnin, A. Brechmann, J. Siegmund, Program comprehension and code complexity metrics: a replication package of an fMRI study, Proceedings - International Conference on Software Engineering (2021) 168–169, https://doi.org/10.1109/ICSE-Companion52605.2021.00071.
[27] T. Busjahn, et al., Eye movements in code reading: relaxing the linear order, in: IEEE International Conference on Program Comprehension, 2015-Augus, 2015, pp. 255–265, https://doi.org/10.1109/ICPC.2015.36.
[28] M. Castelo-branco, I. Cibit, R. Couceiro, A quick review on machine learning techniques in code comprehension and code review estimated by neurophysiological data, in: Proceedings - IEEE 21st Mediterranean Electrotechnical Conference, MELECON 2022, 2022, pp. 408–413.
[29] J. Pfeiffer, B. Rumpe, D. Schmalzing, A. Wortmann, Composition operators for modeling languages: a literature review, Journal of Computer Languages 76 (February) (2023) 101226, https://doi.org/10.1016/j.cola.2023.101226.
[30] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10, https://doi.org/10.1145/2601248.2601268.
[31] E. Mourão, J.F. Pimentel, L. Murta, M. Kalinowski, E. Mendes, C. Wohlin, On the performance of hybrid search strategies for systematic literature reviews in software engineering, Inf. Software Technol. 123 (July 2019) 2020, https://doi.org/10.1016/j.infsof.2020.106294.
[32] N. Gervasoni, A code centric evaluation of C/C ++ vulnerability datasets for deep learning based vulnerability detection techniques, in: Proceedings of the 16th Innovations in Software Engineering Conference, vol. 1, Association for Computing Machinery, 2023, pp. 1–10, https://doi.org/10.1145/3578527.3578530, 1.
[33] Q. Cutts, M. Kallia, Introducing Modelling and Code Comprehension from the First Days of an Introductory Programming Class, ACM International Conference Proceeding Series, 2023, pp. 21–24, https://doi.org/10.1145/3573260.3573266.
[34] C. Schulte, Block Model: an educational model of program comprehension as a tool for a scholarly approach to teaching, in: ICER'08 - Proceedings of the ACM Workshop on International Computing Education Research, 2008, pp. 149–160, https://doi.org/10.1145/1404520.1404535.
[35] L. Silva, A.J. Mendes, A. Gomes, G. Fortes, Investigating programming students problem comprehension ability and its association with learning performance, IEEE Trans. Educ. 66 (2) (2023) 156–162.
[36] D.G. Feitelson, From code complexity metrics to program comprehension, Commun. ACM 66 (5) (2023) 52–61, https://doi.org/10.1145/3546576.
[37] H. Jebnoun, H. Ben Braiek, M.M. Rahman, F. Khomh, The scent of deep learning code: an empirical study, in: Proceedings - 2020 IEEE/ACM 17th International Conference on Mining Software Repositories, MSR 2020, 2020, pp. 420–430, https://doi.org/10.1145/3379597.3387479.
[38] F. Pecorelli, F. Palomba, D. Di Nucci, A. De Lucia, Comparing heuristic and machine learning approaches for metric-based code smell detection, in: IEEE International Conference on Program Comprehension, 2019-May, 2019, pp. 93–104, https://doi.org/10.1109/ICPC.2019.00023.
[39] C.V. Alexandru, S. Panichella, H.C. Gall, Replicating Parser Behavior Using Neural Machine Translation, IEEE International Conference on Program Comprehension, 2017, pp. 316–319, https://doi.org/10.1109/ICPC.2017.11.
[40] C. Politowski, et al., A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension, Inf. Software Technol. 122 (January) (2020), https://doi.org/10.1016/j.infsof.2020.106278.
[41] D.A. Umphress, T.D. Hendrix, J.H. Cross, S. Maghsoodloo, Software visualizations for improving and measuring the comprehensibility of source code, Sci. Comput. Program. 60 (2) (2006) 121–133, https://doi.org/10.1016/j.scico.2005.10.001.
[42] N. Kulkarni, V. Varma, Supporting comprehension of unfamiliar programs by modeling cues, Software Qual. J. 25 (1) (2017) 307–340, https://doi.org/10.1007/s11219-015-9303-5.
[43] J. Mucke, S. Apel, J. Siegmund, Understanding comprehension of iterative and recursive programs with remote eye tracking. Annual Conference on Psychology of Programming Interest Group (PPIG), 2021, pp. 1–17.
[44] J.A.S.A.S. da Costa, R. Gheyi, F. Castor, P.R.F.R.F. de Oliveira, M. Ribeiro, B. Fonseca, Seeing confusion through a new lens: on the impact of atoms of confusion on novices' code comprehension, Empir. Software Eng. 28 (4) (2023), https://doi.org/10.1007/s10664-023-10311-0.
[45] J. Barthélemy, T. Kubo, T.D. Itoh, K. Ikeda, K. Ikeda, Learning to mimic programmers gaze behavior for program comprehension improvement, Artif. Life Robot. 28 (2) (2023) 295–306, https://doi.org/10.1007/s10015-023-00868-w.
[46] R. Southwell, C. Mills, M. Caruso, S.K. D'Mello, Gaze-based predictive models of deep reading comprehension, User Model. User-Adapted Interact. 33 (3) (2023) 687–725, https://doi.org/10.1007/s11257-022-09346-7.
[47] 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: Proceedings - EMIP 2018: Eye Movements in Programming, 2018, pp. 1–5, https://doi.org/10.1145/3216723.3216726.
[48] M.K.C. Yeh, D. Gopstein, Y. Yan, Y. Zhuang, Detecting and comparing brain activity in short program comprehension using EEG, in: Proceedings - Frontiers in Education Conference, FIE, 2017-Octob, 2017, pp. 1–5, https://doi.org/10.1109/FIE.2017.8190486, no. 1444827.
[49] S. Lee, et al., Comparing programming language comprehension between novice and expert programmers using EEG analysis, in: Proceedings - 2016 IEEE 16th International Conference on Bioinformatics and Bioengineering, BIBE 2016, 2016, pp. 350–355, https://doi.org/10.1109/BIBE.2016.30.
[50] I. Crk, T. Kluthe, A. Stefik, Understanding programming expertise: an empirical study of phasic brain wave changes, ACM Trans. Comput. Hum. Interact. 23 (1) (2016), https://doi.org/10.1145/2829945.
[51] S. Lee, D. Hooshyar, H. Ji, K. Nam, H. Lim, Mining biometric data to predict programmer expertise and task difficulty, Cluster Comput. 21 (1) (2018) 1097–1107, https://doi.org/10.1007/s10586-017-0746-2.
[52] 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," International Symposium on Empirical Software Engineering and Measurement, 2018, pp. 1–10, https://doi.org/10.1145/3239235.3240495.
[53] Z. Chentouf, A cognitive system to teach software maintenance project staffing, TEM J. 6 (4) (2017) 699–706, https://doi.org/10.18421/TEM64-08.
[54] O. Elshinnawey, G. Abuguyan, Z. Chentouf, A cognitive attraction network approach to the software team building decision problem, Journal of King Abdulaziz University-Computing and Information Technology Sciences 5 (1) (2016) 75–81, https://doi.org/10.4197/comp.5-1.5.
[55] R. Pekrun, The control-value theory of achievement emotions: assumptions, corollaries, and implications for educational research and practice, Educ. Psychol. Rev. 18 (4) (2006) 315–341, https://doi.org/10.1007/s10648-006-9029-9.
[56] Q. Wang, M. Wang, Y. Yang, X. Zhang, Multi-modal emotion recognition using EEG and speech signals, Comput. Biol. Med. 149 (Oct. 2022) 105907, https://doi.org/10.1016/J.COMPBIOMED.2022.105907.
[57] S. Greipl, et al., When the brain comes into play: neurofunctional correlates of emotions and reward in game-based learning, Comput. Hum. Behav. 125 (January) (2021) 106946, https://doi.org/10.1016/j.chb.2021.106946.
[58] S. Jain, K. Asawa, Modeling of emotion elicitation conditions for a cognitive-emotive architecture, Cognit. Syst. Res. 55 (2019) 60–76, https://doi.org/10.1016/j.cogsys.2018.12.012.
[59] O. Larue, et al., Emotion in the common model of cognition, Procedia Comput. Sci. 145 (2018) 740–746, https://doi.org/10.1016/j.procs.2018.11.045.
[60] L. Rösner, S. Winter, N.C. Krämer, Dangerous minds? Effects of uncivil online comments on aggressive cognitions, emotions, and behavior, Comput. Hum. Behav. 58 (2016) 461–470, https://doi.org/10.1016/j.chb.2016.01.022.
[61] A. Abbad-Andaloussi, T. Sorg, B. Weber, Estimating developers' cognitive load at a fine-grained level using eye-tracking measures, in: IEEE International Conference on Program Comprehension, 2022-March, 2022, pp. 111–121, https://doi.org/10.1145/3524610.3527890.
[62] T. Sorg, A. Abbad-Andaloussi, B. Weber, Towards a fine-grained analysis of cognitive load during program comprehension, in: s, 2022, pp. 748–752, https://doi.org/10.1109/SANER53432.2022.00092.
[63] N. Rizun, A. Revina, V.G. Meister, Analyzing content of tasks in Business Process Management. Blending task execution and organization perspectives, Comput. Ind. 130 (2021) 103463, https://doi.org/10.1016/j.compind.2021.103463.
[64] L. Martin, K. Jaime, F. Ramos, F. Robles, Declarative working memory: a bio-inspired cognitive architecture proposal, Cognit. Syst. Res. 66 (2021) 30–45, https://doi.org/10.1016/j.cogsys.2020.10.014.
[65] M. Lv, H. Liu, W. Zhou, C. Zheng, Efficiency model of micro-course study based on cognitive psychology in the college, Comput. Hum. Behav. 107 (April 2019) (2020) 106027, https://doi.org/10.1016/j.chb.2019.05.024.

[66] J. Jia, X. Yang, R. Zhang, X. Liu, Understanding software developers' cognition in agile requirements engineering, Sci. Comput. Program. 178 (2019) 1–19, https://doi.org/10.1016/j.scico.2019.03.005.
[67] V. Khatri, B.M. Samuel, A.R. Dennis, System 1 and System 2 cognition in the decision to adopt and use a new technology, Inf. Manag. 55 (6) (2018) 709–724, https://doi.org/10.1016/j.im.2018.03.002.
[68] A. Siabdelhadi, A. Chadli, H. Cherroun, A. Ouared, H. Sahraoui, MoTrans-BDI: leveraging the Beliefs-Desires-Intentions agent architecture for collaborative model transformation by example, Journal of Computer Languages 74 (June 2022) (2022) 101174, https://doi.org/10.1016/j.cola.2022.101174.
[69] Y. Sánchez, T. Coma, A. Aguelo, E. Cerezo, ABC-EBDI: an affective framework for BDI agents, Cognit. Syst. Res. 58 (2019) 195–216, https://doi.org/10.1016/j.cogsys.2019.07.002.
[70] J. Selva, What is albert ellis' ABC model in CBT theory?. Positivepsychology.Com, No. May, 2018, pp. 1–26 [Online]. Available: https://positivepsychology.com/albert-ellis-abc-model-rebt-cbt/.
[71] D. Sarracino, G. Dimaggio, R. Ibrahim, R. Popolo, S. Sassaroli, G.M. Ruggiero, When REBT goes difficult: applying ABC-DEF to personality disorders, J. Ration. Emot. Cogn. Behav. Ther. 35 (3) (2017) 278–295, https://doi.org/10.1007/s10942-016-0258-7.
[72] J.D. Kralik, et al., Metacognition for a common model of cognition, Procedia Comput. Sci. 145 (2018) 730–739, https://doi.org/10.1016/j.procs.2018.11.046.
[73] A. Mishra, V. Srivastava, "Cognition based selection and categorization of maintenance engineer (agent) using Artificial Neural Net and Data Mining methods,", in: 2012 CSI 6th International Conference on Software Engineering, CONSEG 2012, 2012, pp. 1–11, https://doi.org/10.1109/CONSEG.2012.6349509.
[74] J. Zupan, Introduction to artificial neural network (ANN) methods : what they are and how to use them, Acta Chim. Slov. 41 (1994) 327–352.
[75] S. Kumar, R.B. Mishra, Cognition based service selection in semantic web service composition, INFOCOMP J. Comput. Sci. 7 (3 SE-Articles) (Sep. 2008) 35–41 [Online]. Available: https://infocomp.dcc.ufla.br/index.php/infocomp/article/view/227.
[76] N. Peitek, J. Siegmund, S. Apel, What Drives the Reading Order of Programmers? an Eye Tracking Study, IEEE International Conference on Program Comprehension, 2020, pp. 342–353, https://doi.org/10.1145/3387904.3389279.