Received August 4, 2021, accepted August 21, 2021, date of publication August 24, 2021, date of current version September 7, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3107795

EEG Activities During Program Comprehension: An Exploration of Cognition

YU-TZU LIN[1,2], (Member, IEEE), YI-ZHI LIAO[1,2], XIAO HU[3], AND CHENG-CHIH WU[1,2]
[1]Advanced Center for the Study of Learning Sciences, National Taiwan Normal University, Taipei 106, Taiwan
[2]Graduate Institute of Information and Computer Education, National Taiwan Normal University, Taipei 106, Taiwan
[3]Human Communication, Development, and Information Sciences, Faculty of Education, The University of Hong Kong, Hong Kong

Corresponding author: Yu-Tzu Lin (linyt@ntnu.edu.tw)

This work was supported in part by the Ministry of Science and Technology of Taiwan under Grant MOST109-2511-H-003-010-MY3.

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the appropriate governmental authorities and organizations under Approval No. NSC101-2511-S-003-037.

ABSTRACT This study attempts to explore cognition during program comprehension through physiological evidence by recording and comparing electroencephalogram (EEG) activities in different frequency bands and the eye movements of the participants with high or low programming abilities. An experiment was conducted with thirty-three undergraduate students majoring in Computer Science. We recorded their EEG activities when they were reading two programs with three types of program constructs. At the same time, the participants' eye movements were recorded by an eye tracker to further understand the relationship between the program comprehension process and EEG activities. Experimental results show that the high-performance participants displayed higher performance for working memory (theta power), attention resource allocation (lower alpha power), and interaction between working memory and semantic memory (upper alpha power) in program comprehension tasks of complex constructs, which proves related theories proposed in the existing research on programming and cognition. The results of this study not only offer objective evidence of the roles cognition plays in program comprehension but also provide educators with suggestions for designing suitable pedagogical strategies.

INDEX TERMS Computer science education, programming, educational technology.

I. INTRODUCTION

Computer programming is a challenging skill involving complicated tasks executed based upon programmers' ***mental models*** based upon cognitive structures (what programmers possess in their memories) and cognitive processes (involved in using or adding knowledge) [1]. Students in computer science make their programming plans and develop specific strategies are based upon the programmers' mental models [2]. The mental model is used to guide information processing between ***working memory*** and the development of production systems in ***long-term memory*** [1], [3]. As illustrated in Fig 1, when a person interacts with a cognitive task, he or she acquires knowledge by combining knowledge structures stored in long-term memory with information in working memory to form a mental model (or representation), used to execute a task. With regard to the execution of computer programming tasks, various mental models influence learners' cognitive processes while writing or comprehending programs which then influence their programming skills [4]. Programming experts build their mental models based upon programming knowledge stored in long-term memory which includes programming structures, rules of programming discourse, and planning knowledge [5], [6]. Experts in computer programming are able to apply their knowledge more effectively than novices [6]. Programming knowledge has been found to mediate the effects of working memory and experience with programming performance [3]. Few studies, however, have examined the relationship between programming skill and its possible antecedents (memory capacity, mental models, and programming strategies) [7]. In addition, existing research relies more on subjective evidence, such as participants' behaviors or results from interviews: consequently, previous studies are unclear with regard to the importance of parameters like memory capacity, mental models, and programming strategies in the overall process of learning

The associate editor coordinating the review of this manuscript and approving it for publication was Saqib Saeed.

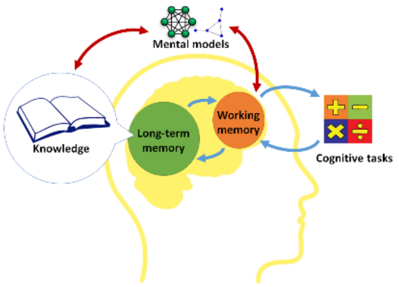


FIGURE 1. The mental model and memory.

and applying skills in the area of computer programming. Further research will be needed in order to clarify and explain these relationships.

The rapid development of ***neuroscience research*** provides more possibilities to uncover the underlying mechanisms of information processing in the brain during computer programming [8], [9]. Knowledge stored in long-term memory and knowledge used in working-memory have previously been found to be good predictors of computer programming skill acquisition in traditional research [10]–[12], which has been confirmed by recent research in neuroscience [13], [14]. Existing research, however, focuses more upon mapping neuroscientific data to the level of programming skill but often fails to provide explanations about the cognitive meanings or specific brain mappings. Many neuroscientific studies have focused upon specific neurologically-active chemicals, yet their effects upon cognitive meanings (brain maps) are often unclear [15], [16]. Integrating the results from different neurologic and physiologic methods and procedures might help provide more accurate explanations with regard to cognition [17]. ***Electroencephalogram*** (EEG) has higher temporal resolution for illustrating cognitive dynamics during problem solving. Consequently, we chose to measure brain activity using EEG which has been successfully used in previous existing cognitive research [15]. In the field of computer programming, a paucity of research exists studying the relationship of EEGs with respect to the cognition of programming dynamics [8], [9]. Although the possible relationship between the memory and programming skill has been depicted in previous research, the use of EEGs in this research is a relatively novel way to explore programming cognition, supplemented by eye tracking to reveal more about how different cognitive models influence programming dynamics and skills.

The participants' EEG activities were recorded while students were engaged in understanding a specific computer program. In order to align EEG activities with the visual program statements, an eye tracker was employed to record eye-tracking paths during program comprehension. Triangulating EEG activities and eye movements helped to elucidate cognitive processes involved in program comprehension. Previous research has addressed the functional significance of theta and alpha frequency bands in programming comprehension [8], [9], cognitive activities such as problem solving [18], working-memory operation [19], [20], and semantic understanding [21], which have shown to be involved in computer programming. The present research focuses on theta and alpha bands and their involvement with programming skills.

A list of our research questions (labeled R) and our hypotheses (labeled H) follow:

R1: What are the differences in EEG activities during program comprehension between high- and low-performing participants?

i. What differences exist for different types of program constructs?

ii. What differences exist with regard to more complex, difficult programming functions, tasks?

Based upon the existing research on programming comprehension, four hypotheses, designated H1.1 through H1.4 were developed in order to address EEG results for research question one, designated R1, which appear as follows:

The first hypothesis is based on the previous findings arguing that high-performance programmers have been shown to display higher performance with regard to working memory needed for mental calculations as well as better cue-based search strategies required for programming comprehension [9], [22].

H1.1: High-performance participants are expected to have stronger theta wave power than low-performance participants.

Because program comprehension tasks require more attention resources leading to higher comprehension and performance, our second hypothesis is as follows:

H1.2: High-performance participants will have stronger *lower alpha power* waves than low-performance participants.

Because high-performance programmers master programming knowledge compilation, which requires frequent interaction between the working and the semantic memory, our third hypothesis is as follows:

H1.3: High-performance participants have a stronger *upper alpha power waves* than low-performance participants.

With more complex programming constructs or different programming functions, participants would need higher cognitive abilities: thus, the differences in EEG activities should be more obvious. This lead to the formulation of our 4th hypothesis:

H1.4: The difference in EEG activities between high- and the low-performance participants should be more obvious when involving complex constructs or more difficult programming functions.

The second research question addresses implications derived based on the cognitive meanings of the EEG activities, and appears as follows:

R2: What do the EEG activities during program comprehension processes tell us about the cognitive factors that might affect program comprehension?

Answers to the above research questions appear in our data analysis and discussion sections.

II. LITERATURE REVIEW

A. PROGRAM COMPREHENSION

Program comprehension plays an important role in computer programming, which is the foundation of adding to or modifying the functionality of programs [23]. Therefore, some research has tried to analyze how programmers comprehend programs by citing theoretical frameworks and methodologies from related areas, such as text comprehension, problem solving, and education [24], [25]. Brooks initialized the studies of theoretical models for program comprehension [10]. He argued that program comprehension is hypothesis-driven and is a top-down process [26]. Programming problem solving is directed by a production system which is a type of internal control mechanism that consists of a set of conditions and actions to be performed during programming [10]. Soloway and Ehrlich believed that programs are executable and communicative entities [6]. Research findings in text reading and comprehension have shown that the goal of reading a program necessitates the formulation of specific plans. These plans consist of program fragments representing stereotypical action sequences in programing. Using empirical evidence to illustrate, experts have used a top-down approach in order to decompose programming goals and plans into lower-level plans and program codes. Pennington developed a mental model for program comprehension also based on text comprehension [5]. Pennington's model utilized four types of abstraction implied in program texts: (1) functional, (2) data flow, (3) control flow, and (4) conditional actions. In Pennington's study, an experiment was conducted to examine experts' program comprehension. The results show that programming *experts* formed their mental representations of programming based mostly on control flow rather than functional or data flow abstraction. Wiedenbeck and Ramalingam summarized Pennington's study, categorizing control flow and basic operations as the ***program model***, and data flow and functions as the ***domain model*** [23]. The program model contains information about program entities and relationships in the program text (e.g., the data structure and the code execution sequence). The domain model, on the other hand, is concerned with how data passes through a program, and the transformation of data from the input state to the output state. In order to understand the differences in program comprehension under different paradigms (object-oriented and process-oriented), investigators conducted a similar experiment. Their results showed that the process-oriented paradigm resulted in a higher precision rate in the *program* model. The object-oriented one, however, showed better performance in the *domain* model.

In spite of the previously cited findings, other studies by Pennington, Wiedenbeck, and others have shown that both the *program* and *domain* mental models were of equal importance for program comprehension [23]. A study by Letovsky used verbal protocol analysis to collect experts' cognitive processes [26]. Based on the collected data, the author proposed a computational model of program comprehension in which programmers used their programming knowledge to understand programs, ask questions, and to predict programming behaviors. They were then able to find and verify answers from the programs. This model consists of a combination of assimilation processes used to construct cognitive models, during which programmers would be "opportunistic" in using both the top-down and bottom-up approaches to help construct models for understanding the target programs. Table 1 summarizes the program comprehension models proposed by various scholars.

TABLE 1. Studies on program comprehension models.

Authors (# of citations)	Year	Assimilation	Cognition
Brooks (835)	1983	top-down	Program comprehension is a top-down and hypothesis-driven reconstruction process. The hypothesis is refined and elaborated based on information extracted from the code.
Soloway & Ehrlich (965)	1984	top-down	Program comprehension involves composing the programming plans that have been modified to fit the needs of specific problems.
Pennington (625)	1987	bottom-up	Program comprehension involves detecting or inferring different kinds of relations between program parts.
Letovsky (437)	1987	opportunistic	Programmers utilize their knowledge base of computer programs for understanding the tasks, constructing mental models for encoding current understanding, and use an assimilation process for interacting with the stimulus materials.

One study examined the above models and suggested that most of the program comprehension models use elements such as external representation, cognitive structure and assimilation process [27]. External representation is formed from any materials and data associated with the target program but not a part of the internal knowledge of the programmers. Cognitive structure constitutes the internal knowledge of the programmers, which can be divided into their knowledge base and mental representations. The knowledge base includes the general programming knowledge of the programmers and knowledge associated with that area. The formation of mental representations is a process during which programmers construct internal knowledge from the target program using the knowledge base as they acquire program comprehension. Lastly, the assimilation process is related to specific strategies adopted by programmers. These strategies extract information from source codes to allow programmers to construct a mental representation of the program code while enabling them to comprehend the program.

Several studies attempted to investigate the formation of mental representations during program comprehension and attempted to examine the differences among expert and novice learners [23], [28]–[30]. With recent advances in cognitive neuroscience and the advancement in more skilled equipment, some studies began to use eye trackers to study cognitive processes during program comprehension. Studies conducted by Crosby *et al.* [31] and Aschwanden and Crosby

[32], for example, showed certain important aspects of computer programming exhibit significant differences between high- and low-performance participants as detected by the frequency of eye fixations. Investigators have found that certain key statements in computer problems, called "*beacons*," play a key role in program comprehension. These "beacons" may consist of indicate specific structures or statements in computer programs vital for proper program comprehension. At the same time, a beacon provides a connection between programs and the process for establishing hypotheses. This connection helps programmers verify their hypotheses about the goals of the programs [33]. Crosby *et al.* [31] found that more experienced programmers would focus more on important program codes, meaning they knew how to use beacons. In contrast, novice learners seldom used beacons in their process of program comprehension. From the perspective of program structure, studies based on eye-tracking data revealed that cognitive behaviors differed in comprehending programs with and without recursions [32]. Investigators have also shown that the use of "recursions" is also an important concept and a foundational technique for problem-solving in computer science. Dicheva and Close [34] reported, however, that novice learners tended to comprehend recursions using the concept of "loop", while experts employed the "stack" concept. This demonstrated that the two groups had constructed different mental models during their programming experiences.

B. PROGRAM COMPREHENSION AND NEUROSCIENCE

Most traditional research with regard to the learning of computer programming has utilized quizzes, questionnaire surveys [5], [23], and "thinking aloud" protocols [26], [28], along with the use of semi-structured interviews to learn about participants' cognition during programming. The aforesaid methods still, unfortunately, rely on researchers' inferences with regard to the participants' cognitive development. Consequently, prior analyses might not be as objective, precise, or comprehensive. Recent advances in equipment and skills used to investigate cognitive processes in neuroscience now allow researchers to explore more fully the contours of cognition with more accurate physiological evidence. Neuroscience techniques such as Positron Emission Tomography (PET), functional Magnetic Resonance Imaging (fMRI), EEG, and so on allow for more detailed of brain activity when subjects are engaged in cognitive activities.

Some efforts have been made to link brain activity with the acquisition of computer programming skills. Some research has examined brain activities during code comprehension, code review, and prose review based on fMRIs and shown differences between neural representations in programming languages and natural languages [14]. Programming languages may be akin to natural languages with greater expertise. The fMRI technique has also been employed to study activated brain regions related to working-memory operations, attention, and language processing while comprehending the programs in a bottom-up manner [35]. Another study addressed the differences in activated brain regions between top-down and bottom-up comprehension [36]. Investigators have shown that "bug" suspicion and bug detection also involve different brain activities [37]. Functional MRIs provides better spatial resolution for brain activities, but lacks temporal dynamics. For this reasons, the techniques of EEGs and functional Near-Infrared Spectroscopy (fNIRS), have been employed in other studies. Research has shown that one's cognitive load during comprehension can be predicted based upon the techniques of EEG activity, cerebral blood flow, and even oxygenated hemoglobin (oxy-Hb) [38], [39]. Frontal lobe activity has been shown to be a useful method for quantifying the workload in short-term memory during program comprehension tasks, as well as increased activity when performing different tasks. Investigators have shown a significant difference between variable manipulations, numeric calculation, and conditional statements based on fNIRS [40]. The presence of linguistic anti-patterns in computer programs increases the cognitive load during program comprehension tasks, but structural and readability characteristics do not [41]. Some additional studies have employed EEGs to explore program comprehension: alpha and theta waves were used to quantify programmers' expertise and detect the difficulty of program comprehension tasks [8], [9]. These two bands have frequently been used in the research of the following: problem solving [18], working-memory operations [19], and semantic understanding [21], which are likely involved with the acquisition of computer programming skills. Another study by Lee, *et al.* depicted the difference between experts and novices based on beta and gamma powers and found experts to be superior with the tasks of digit encoding, coarse coding, short-term memory, and subsequent memory effects [9]. When compared with syntax tasks, higher brain activation was found during comprehension tasks in the theta band. These findings imply that searching for rules for syntax tasks is easier than that of code comprehension, which requires the mental simulation of code execution [38]. EEG has a higher temporal resolution and can reveal cognitive dynamics as evidenced by varying activities of specific EEG wave bands. The use of EEGs when investigating computer programming dynamics, has, however, been somewhat limited and much work is needed to elucidate the nature of brain activity as correlated with the formation of mental models and programming strategies used by programmers.

C. EEG ACTIVITIES AND COGNITION

When a specific cognitive task is being performed, activities of corresponding parts of the brain have been shown to increase. In order to explore how brain activities are associated with cognitive activities, neuroscience techniques such as PET and fMRI scans are employed. These techniques are useful, but cannot describe the dynamic process of cognition [42]. In order to describe more dynamic processes, EEG waveforms have been examined to describe the temporal variations of brain waves [43], [44]. This type of research linking human EEG activities to cognition has been growing rapidly and helps to explain the role of neurophysiology in cognition [18]. The use of EEGs has been extensively utilized in the areas of psychology and education with regard to understanding various processes of student learning. Previous research has shown the functional significance of *theta* and *alpha* band frequencies with the execution of complex tasks–those which involve strategy selection and execution [18], [45]. It is unclear from these few studies, however, as to the exact role of sequential modulations involved in strategy selection and execution [15].

Theta waves have been associated with many cognitive processes related to one's working memory [19], [46]–[48], especially when new information is encoded into episodic memory [21]. Klimesch *et al.* [48] tested this hypothesis and found that the participants who remembered the words displayed a higher theta power than those who could not remember them. Similar results were obtained in the virtual maze experiment in two studies conducted by Kahana *et al.* [49] and Caplan *et al.* [50]. Their results showed that when exploring the maze, theta waves were more prominent. The use of longer mazes showed longer times for theta wave vibration. Some studies have claimed that theta wave are also associated with one's working-memory load. Gevins *et al.* [19] conducted the N-back experiment and discovered that increasing task difficulty would also raise loads involved with working-memory, and also showed the theta wave power to be increased [19]. Research has also shown that problem-solving processes definitely involve theta waves (increasing their activity). Students who were able to find the best solutions showed a significantly higher theta activity than other less-skilled students [51]–[53]. The activity of the theta band has been shown to increase parametrically with the number of items retained in the working memory [54] and is relatively specific for control of the working memory. The theta wave has also been shown to be important for executive functions in one's working memory [13].

It is interesting to note that ***alpha*** and *theta* waves have been shown to be closely associated. Many studies have shown that alpha and theta waves are both associated with working memory [53], [55]. Alpha power increases mainly during the retention of visual or verbal material in the working memory [56]. Alpha waves, however, seem to be are more related to memory retrieval [57], [58]. The alpha wave is also associated with the rate of information processing and memory performance. In order to identify more specific differences in alpha wave functions, many studies [57], [59], [60] have used narrower frequency bands such as *lower alpha* and *upper alpha* bands instead of the entire alpha band. The ***lower alpha band*** (8.3 ± 10.3 Hz) reflects attentional processes, whereas the ***upper alpha band*** (10.3 ± 12.3 Hz) reflects stimulus-related cognitive processes [21]. An increase of the lower alpha power reflects an attempt to increase attention, whereas a large upper alpha power indicates good cognitive performance. A lower alpha power is related to the allocation of attentional resources in the human somatosensory system, and the upper alpha is related to cognition, involving the interaction between the working and long-term memories. Klimesch [58] proposed that the upper alpha wave reflects the encoding and processing of the semantic memory [58]. A stronger upper alpha power indicates good performance on both coordinating the cognitive process and semantic understanding. In addition to the processing of the semantic memory, Klimesch *et al.* [55] suggested the upper alpha band plays an important role in long-term memory and is related to the reactivation of long-term memory codes in short-term memory. The alpha power is also related to "top-down" processing [61]. The alpha power increases during retention of information and top-down processing for the performance of upcoming tasks in one's working memory [62], [63]. Top-down (concept-driven) processing occurs when the perceptual representation is influenced by cognitive intention (e.g., previous knowledge, expectation, active redirection of attention, and mental readiness), which can increase the speed and efficiency of perceptual identification [63]. In addition to the memory function, the alpha wave is associated with mental effort and intelligence. Highly intelligent participants display a higher alpha power [64], [60]. These results can be explained by the fact that highly intelligent participants spent less mental effort when solving problems than participants with average intelligence. This low mental effort and higher intelligence is shown by a high alpha power brain wave. This result can be explained by the "brain efficiency" theory proposed by Haier *et al.* [65]. They argued intelligence meant how efficiently the brain operated. Efficient operation means using the brain lobes related to specific tasks but not those irrelevant to the tasks. Therefore, studies have concluded the alpha wave responds to the inhibition of the cortex irrelevant to the tasks [66], [67].

III. METHODOLOGY

A. PARTICIPANTS

This study targeted undergraduate students with backgrounds in computer science and consisted of students willing to participate in the study and employed the method of "convenience sampling." Requirements for participants and information on the experiment were announced on a popular bulletin board system for college students in order to attract participants. In the subject recruitment notice, details of the experimental purpose and procedure, equipment utilized, time required, and payment were provided. Students who were willing to participate in the study replied and selected suitable time slots for the experiment. The volunteers for the experiment understood the details of the experiment and had signed the participant consent form. Data from the investigation were analyzed anonymously to protect the confidentiality of participants. The research plan had been reviewed and approved by the appropriate governmental authorities and organizations. Two participants were considered to be invalid and excluded from the data analysis because their computers crashed during the experiment and their EEG data was incomplete. The number of participants, thus, was 33 and consisted of some 20 males and 13 females. All the participants had received at least one year of education in C programming. To ensure the EEG activities of the participants would not be affected by physiological variations or mental abnormalities, the participants were all right-handed and did not have a history of psychological disorder or mental illness. In addition, since this study tracked eye movements, participants with the following conditions were excluded: (1) wearing hard contact lenses, (2) wearing cornea color film or fake eyelashes, (3) having eye problems such as strabismus, and (4) receiving eye therapy and medical treatment.

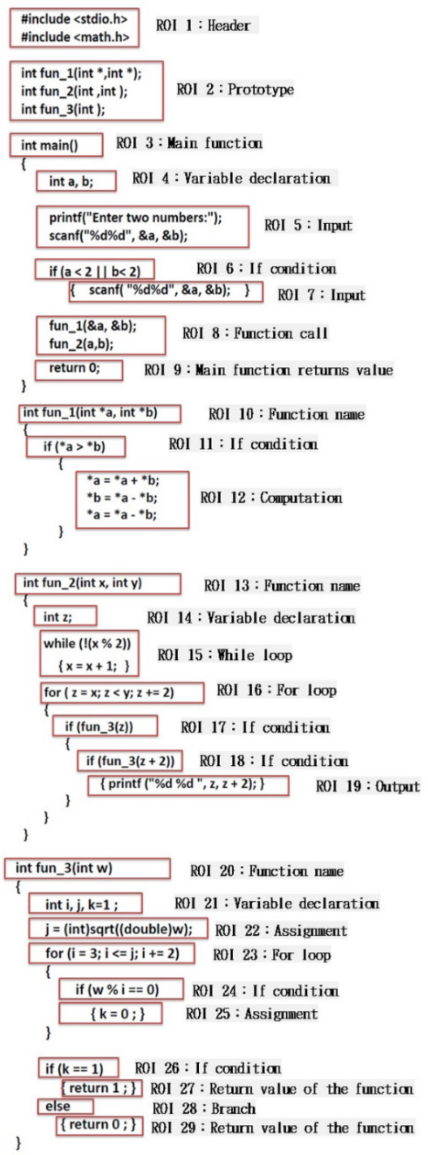
B. STIMULI AND APPARATUS

The stimulus material included two program problems written in C programming language. Each problem included three functions. The first problem involved iteration, in which the program goal was to print out twin primes (i.e., two adjacent prime numbers). The second problem used mainly a recursion structure, in which the program goal was for conversion between numerical systems. Specifically, the program was asked to convert decimal numbers first to binary ones, then to octal.

This study used the NeuroSky MindWave headset to record EEG activities. This headset has been employed in many cognitive [68] and multidisciplinary [69]–[71] studies, and has been proven to be comparable to medical-grade EEG equipment [72], [73]. It has also been proven to be valid by comparing results using the aforesaid equipment with existing cognitive tests [74]. In the present study, the accurate recording of eye movements was aligned with EEG activities to confirm complex cognitive processes. The NeuroSky MindWave headset is also portable and was very well suited for our experimental setting. The headset used for the study consisted of a customer-grade device but also included noise filtering for EMGs (electromyography). It can, therefore, remove possible influences caused from eye blinks.

Computer program comprehension is more complex than many other cognitive tasks, and required a longer amount of time to study and acquire data. In addition, to delineate the detailed cognitive processes and logic involved, the programs in the experiment were sufficiently long. Consequently, the large data sets eliminated errors and possible noise of the EEG signals. The EEG headset used in our study had a sampling rate of 512 Hz. It measures EEG waves by using dry electrodes located on the forehead (FP1) and by reference electrodes placed at the earlobes. It provided information from eight frequency bands: delta (1-3 Hz), theta (4-7 Hz), lower alpha (8-9 Hz), upper alpha (10-12 Hz), lower beta (13-17 Hz), upper beta (18-30 Hz), lower gamma (31-40 Hz) and upper gamma (41-50 Hz). To explain the research results more accurately, only theta, lower alpha, and upper alpha waves, which are associated with the relevant cognition during program comprehension, were discussed in this study.

In addition to EEG data, eye tracking was also employed to help explain cognitive processes [75], [76]. A Tobii X120 eye



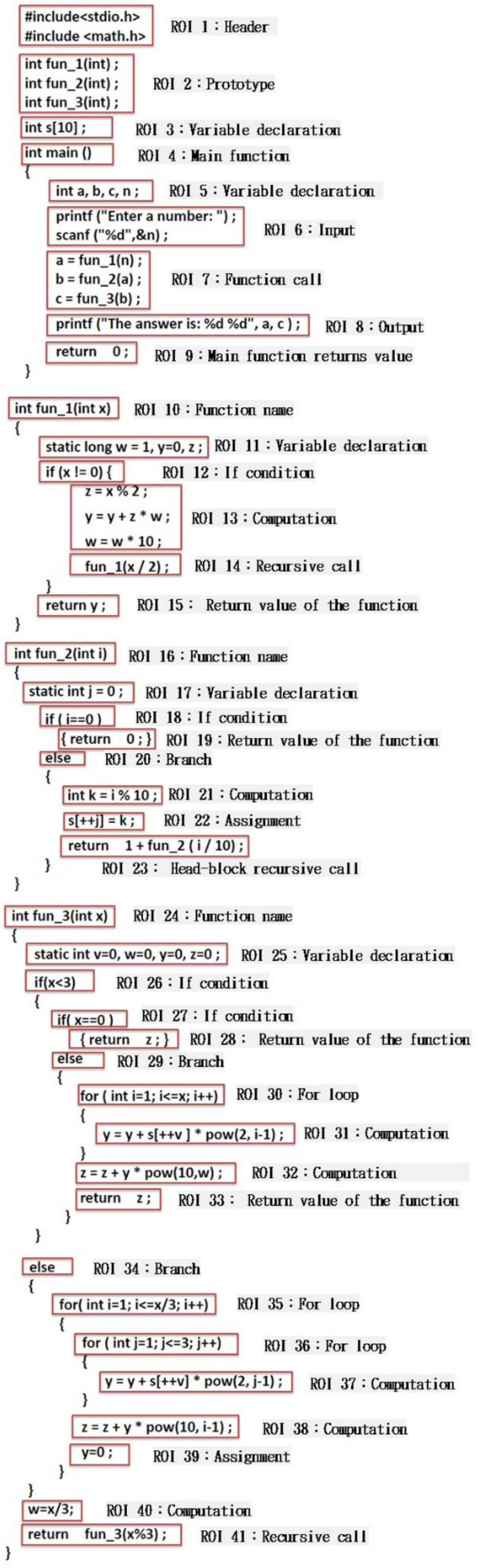
```
#include <stdio.h>                 ROI 1：Header
#include <math.h>

int fun_1(int *,int *);            ROI 2：Prototype
int fun_2(int ,int );
int fun_3(int );

int main()                         ROI 3：Main function
{
    int a, b;                      ROI 4：Variable declaration

    printf("Enter two numbers:");  ROI 5：Input
    scanf("%d%d", &a, &b);

    if (a < 2 || b< 2)             ROI 6：If condition
        {   scanf( "%d%d", &a, &b);   }   ROI 7：Input

    fun_1(&a, &b);                 ROI 8：Function call
    fun_2(a,b);
    return 0;                      ROI 9：Main function returns value
}
int fun_1(int *a, int *b)          ROI 10：Function name
{
  if (*a > *b)                     ROI 11：If condition
     {
        *a = *a + *b;              ROI 12：Computation
        *b = *a - *b;
        *a = *a - *b;
     }
}

int fun_2(int x, int y)            ROI 13：Function name
{
   int z;                          ROI 14：Variable declaration
   while (!(x % 2))                ROI 15：While loop
     {x = x + 1; }
   for ( z = x; z < y; z += 2)     ROI 16：For loop
   {
     if (fun_3(z))                 ROI 17：If condition
     {
       if (fun_3(z + 2))           ROI 18：If condition
         { printf ("%d %d ", z, z + 2); }   ROI 19：Output
     }
   }
}

int fun_3(int w)                   ROI 20：Function name
{
   int i, j, k=1 ;                 ROI 21：Variable declaration
   j = (int)sqrt((double)w);       ROI 22：Assignment
   for (i = 3; i <= j; i += 2)     ROI 23：For loop
   {
      if (w % i == 0)              ROI 24：If condition
         { k = 0 ; }               ROI 25：Assignment
   }

   if (k == 1)                     ROI 26：If condition
     { return 1 ; }                ROI 27：Return value of the function
   else                            ROI 28：Branch
     { return 0 ; }                ROI 29：Return value of the function
}
```

FIGURE 2. Problem 1: program with iteration.

tracker with a sampling rate of 120 Hz was used for tracking the process of eye movements. The distance between the eye tracker and the screen was 15.5 cm. The size of the screen was 52 cm × 29 cm. The eye tracker was at a 35° angle to the screen. The height from the screen to the table was 5.5 cm. The ***region of interest*** (***ROI***) used in the eye movement analysis was defined according to the roles of the program code (such as header file, function prototype and expression, etc.), as shown in Figs 2 and 3. In addition to ROIs, Figs 2 and 3 show that each problem contains three functions, referred to as functions #1, #2 and #3 in this paper.

IV. EXPERIMENTAL PROCEDURE

Before the experiment started, the participants were given instructions about the flow of the experiment. This was followed by calibration of the eye-tracker device. Before starting the experiment, electrodes and skin were ensured to



```
#include<stdio.h>
#include <math.h>                      ROI 1 : Header
int fun_1(int) ;
int fun_2(int) ;                       ROI 2 : Prototype
int fun_3(int) ;
int s[10] ;                            ROI 3 : Variable declaration
int main ()                            ROI 4 : Main function
{
    int a, b, c, n ;                   ROI 5 : Variable declaration
    printf ("Enter a number: ") ;
    scanf ("%d",&n) ;                  ROI 6 : Input
    a = fun_1(n) ;
    b = fun_2(a) ;                     ROI 7 : Function call
    c = fun_3(b) ;
    printf ("The answer is: %d %d", a, c ) ;   ROI 8 : Output
    return   0 ;                       ROI 9 : Main function returns value
}

int fun_1(int x)                       ROI 10 : Function name
{
    static long w = 1, y=0, z ;        ROI 11 : Variable declaration
    if (x != 0) {                      ROI 12 : If condition
        z = x % 2 ;
        y = y + z * w ;                ROI 13 : Computation
        w = w * 10 ;
        fun_1(x / 2) ;                 ROI 14 : Recursive call
    }
    return y ;                         ROI 15 : Return value of the function
}

int fun_2(int i)                       ROI 16 : Function name
{
    static int j = 0 ;                 ROI 17 : Variable declaration
    if ( i==0 )                        ROI 18 : If condition
        { return   0 ; }               ROI 19 : Return value of the function
    else                               ROI 20 : Branch
    {
        int k = i % 10 ;               ROI 21 : Computation
        s[++j] = k ;                   ROI 22 : Assignment
        return   1 + fun_2 ( i / 10) ;
    }                                  ROI 23 : Head-block recursive call
}

int fun_3(int x)                       ROI 24 : Function name
{
    static int v=0, w=0, y=0, z=0 ;    ROI 25 : Variable declaration
    if(x<3)                            ROI 26 : If condition
    {
        if( x==0 )                     ROI 27 : If condition
            { return   z ; }           ROI 28 : Return value of the function
        else                           ROI 29 : Branch
        {
            for ( int i=1; i<=x; i++)  ROI 30 : For loop
            {
                y = y + s[++v ] * pow(2, i-1) ;   ROI 31 : Computation
            }
            z = z + y * pow(10,w) ;    ROI 32 : Computation
            return   z ;               ROI 33 : Return value of the function
        }
    }

    else                               ROI 34 : Branch
    {
        for( int i=1; i<=x/3; i++)     ROI 35 : For loop
        {
            for ( int j=1; j<=3; j++)  ROI 36 : For loop
            {
                y = y + s[++v] * pow(2, j-1) ;   ROI 37 : Computation
            }
            z = z + y * pow(10, i-1) ; ROI 38 : Computation
            y=0 ;                      ROI 39 : Assignment
        }
    }
    w=x/3;                             ROI 40 : Computation
    return   fun_3(x%3) ;              ROI 41 : Recursive call
}
```

FIGURE 3. Problem 2: program with recursion.

be in close contact and the signal reception was tested to make sure the equipment was working properly. The participants were also asked to try their best in comprehending the programs and were told that their eyes should not wander beyond the screen. During the experiment, both EEG activities and eye movements were recorded. After completing each experimental problem, the participants completed a questionnaire on how they comprehended the programs (the goal, logic, and output), and attended an interview to reflect their perceptions and experiences. The participants were then informed that the questionnaire should be completed based on their understanding of the programs during the experiment. For each participant, the experimental duration was about one hour, including the time for comprehending each program (five minutes per program), filling out the comprehension test and the questionnaire (fifteen minutes per program), and the interview. The entire flow of the experiment is presented in Fig 4.

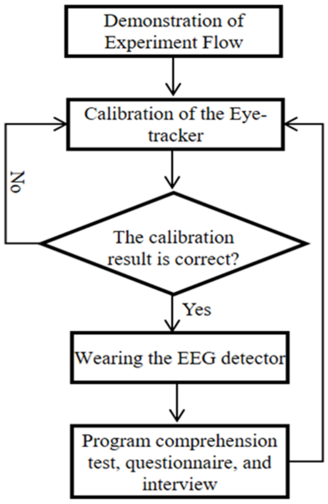

Demonstration of Experiment Flow
Calibration of the Eye-tracker
No
The calibration result is correct?
Yes
Wearing the EEG detector
Program comprehension test, questionnaire, and interview

FIGURE 4. Experimental procedure.

V. DATA COLLECTION AND ANALYSIS

Data collected and analyzed in this study include EEG power and eye-gaze data. The eye-gaze positions had been transferred to ROIs (such as the program header region, the function prototype region, and the variable declaration region) by the eye tracker package (if the eye-gaze position locates in an ROI, then this point is labelled as this ROI). In the pre-processing stage, the EEG data (EEG powers) and eye-gaze data (eye-gaze ROIs) were both down-sampled to 1 Hz. Then the gaze data and the EEG data were aligned in time, which enabled identification of the EEG data when the participants were looking at the programs. Since EEG data gave us only the time series of EEG power without information about attention positions, the eye tracker was employed to locate the attention positions by synchronizing the EEG data with the eye-tracking data. Next, the attention position at the programs for each EEG signal was located by the eye-tracking data. The ROIs for EEG signals were obtained, based on which the EEG power for each ROI could be derived by averaging the EEG powers occurring in the ROI. The EEG powers for each group and program structure/function were derived by averaging the powers of all subjects in each group for comprehending ROIs involved in each program structure/function. Afterwards, the EEG power of the high- and low-performance participants were compared using *Kruskal-Wallis statistical test*ing. The nonparametric test was employed because the sample sizes in the experiment are small.

In this study, participants' performance in program comprehension for the two problems was measured by using a program comprehension test (16 points for the iteration problem and 12 for the recursion problem). Based on the test scores, the participants were grouped into high-performance participants (those with scores above the average) and the low-performance participants (those below-average). Each question on the test was awarded 2 points for a correct answer: "1" for a partially correct answer, and "0" for an incorrect one. The high- (higher or equal to the average) and low-performance participants (below-average) were distinguished based on their scores on the program comprehension test.

The programming constructs involved in the test programs included *condition, iteration and recursion* (as shown in Table 2). For the construct of recursion, Götschi *et al.* [34] divided recursive programs into three types: (1) *head-block recursion*—having another coefficient before the recursive call, e.g., n*fact($n-1$); (2) *embedded recursion*—having a coefficient before and after the recursive call, e.g., 4*prog(n/2) + 3; and (3) *two recursive call*—consisting of two recursive calls, e.g., fib($n-1$) + fib($n-2$). Our program involved a simple *head-block recursion,* e.g., 1 + fun_2 (n /10) as well as another type of recursion without additional terms. We refer to the latter as a *general recursive call,* e.g., fun_1(n/2), in comparison with the *head-block recursion.*

In order to explore why students had difficulty comprehending especially difficult functions, additional methods were needed—other than just the use of interviews and questionnaires. The technique of *sequential analyses* was utilized with eye movements to compare program-tracing sequences between the high- and low-performance participants [77]. The probability of the occurrence of an ROI sequence ROIi→ROIj is significantly high if the z-score of the sequential analysis is larger than or equal to 1.96 (95% confidence level). The z-score of sequence $e_i \rightarrow e_j$, was computed by:

$$z = \frac{f\left(e_ie_j\right)_{obs} - f\left(e_ie_j\right)_{exp}}{\sqrt{N_S \cdot p\left(e_ie_j\right)_{exp} \cdot \left(1 - p\left(e_ie_j\right)_{exp}\right)}}, \quad (1)$$

where $f(e_ie_j)_{obs}$ and $f(e_ie_j)_{exp}$ are the observed and expected values of the frequency of event sequence $e_i \rightarrow e_j$, respectively; $p(e_ie_j)_{obs}$ and $p(e_ie_j)_{exp}$ are the observed and expected values of the probability of event sequence $e_i \rightarrow e_j$, respectively; and N_S is the number of event sequences.

VI. RESULTS AND DISCUSSION

The experiment included thirty-three trials for the thirty-three participants. For each trial, the participants had to comprehend two programs with three functions containing three types of constructs. Descriptive statistics of the test scores are shown in Table 3. Table 4 displays the number of participants (high- and low-performance participants were denoted by H and L, respectively) who correctly answered questions as well as the response time related to each problem and program functions. In addition, a questionnaire and an interview were conducted to understand how the students comprehended the programs.

The following two subsections discuss the results of the differences (examined by the Kruskal-Wallis tests) between the high- and low-performance participants for three program constructs and difficult program functions. The non-parametric Leven test was also performed to guarantee the homoscedasticity of the distributions of the groups [78]. Since we conducted many tests on the same data set, multiple comparison corrections were conducted using the Benjamini-Hochberg procedure in order to counteract the multiple-comparison problem [79]. After the correction, all p values less than the critical value 0.036 were considered significant for the whole confidence level of 0.95.

TABLE 2. Categories of program constructs involved in this study.

Construct	Sub construct	Example

Condition	If statement	if (k == 1)
		if (fun_3(z))

Iteration	For loop, While loop	for (i=3; i<= j; i +=2)
		while (!(x % 2))

Recursion	General recursive call, Head-block recursive call	fun_1(n /2) 1 + fun_2 (n/10)

TABLE 3. Descriptive statistics of the comprehension test results for the high and low-performance participants.

Subjects / Problem	High-performance			Low-performance			All		
	N	Mean	SD	N	Mean	SD	N	Mean	SD
Iteration	18	15.33	1.40	15	4.93	2.30	33	10.61	5.50
Recursion	20	10.20	2.10	13	1.77	1.310	33	6.88	4.50

TABLE 4. Number of participants who correctly answered test questions (total number: # of high performers # of low performers) and average response time (in brackets) related to each problem and program function.

Function / Problem	#1	#2	#3
Iteration	22: 16H 6L (55.63 s)	15: 15H 0L (175.06 s)	19: 18H 1L (164.27 s)
Recursion	20: 20H 0L (311.97 s)	20: 19H 1L (152.06 s)	11: 11H 0L (313.12 s)

A. DIFFERENCES BETWEEN THE HIGH- AND LOW-PERFORMANCE PARTICIPANTS OF DIFFERENT PROGRAM CONSTRUCTS (R1i, R2)

Table 5 shows the descriptive statistics of EEG powers for theta, lower alpha, and upper alpha bands ($\bar{x}$ is the mean of all

EEG powers extracted during comprehending each construct and SD is the standard deviation) for each program construct and the differences between the high- and low-performance participants examined by Kruskal-Wallis tests. In terms of *conditions* and complex *iterations* or *recursions*, the high- and low-performance participants showed significant differences in the theta, lower alpha and upper alpha powers. This finding implies that the high-performance participants possessed a better performance on working-memory operation (theta power), interaction between the working and semantic memories (upper alpha power), and attention ability (lower alpha power). In summary, for the logically complex functions, hypotheses H1.2, H1.1, and H1.3 (as listed in the *Introduction* section) are supported for all program constructs, which also supports for H1.4.

The results of the complex *iterations* and *recursions* are in agreement with existing research: Since the iterative structures and recursive functions are larger, they require substantial memory ability and attention resources to process loop logic, arguments passing, the recursive operations, etc. Low-performance participants struggled more with complex iterative or recursive computations, needed to go back to access previous information, and required additional tools to make more calculation—in order to compensate for the overload of their working memory [22]. The high-performance participants, on the other hand, could recognize beacons, and organize code into chunks in order to reduce their working-memory load [31]–[33]. In addition, the high-performance participants could grasp the *gestault* or *whole logic* by compiling the meaning of the code from the semantic memory in order to construct their mental models. They could, then build plans to conduct top-down attention and decompose the goal of the program into code [6]. This might allow for the more efficient use of attention resources and a better interaction between working and semantic memories. Previous research about reading found that high-performance readers used top-down processing to facilitate their on-going word recognition, but low-performance readers failed to do so [80]. This finding was based on the assumption that comprehension of meaning precedes word identification. Skilled readers extract various featural information and rearrange the featural specifications for meaningful identification, which is similar to what the skilled programmers do: recognize beacons and use them to reorganize the structure for conducting their comprehension plans [31]. The *while-loop iterative construct* in the experiment, however, showed no difference between the two groups of students–probably because the statement of the *while-loop* in the given problem was relatively simple. This is to say that the condition for judging whether the loop continues is (!(x%2)), and the loop was only one line of code. Consequently, the results in memory-related or attention aspects do not show a significant difference when compared with the low-performance participants. The result for *head-block recursion* was found to be similar. A possible explanation for this finding is that although *head-block recursion* frames *recursive calls* in a statement, the *head-block recursive function* in the given problem is actually simpler than the *general recursive function*.

The results of the conditional construct also agree with previous research. Comprehending the *conditional* construct requires computations in working memory [22], while a lack of knowledge on the control construct (programming plans or rules of programming discourse) results in an overload of the working memory, which, in turn affects the process of problem solving [81]. Therefore, the high-performance participants displayed better working-memory performance (stronger theta power) when computing the conditional statements which could compile the meaning of the code based on the patterns in the semantic memory (stronger upper alpha power [21]) which are involved with conditional logic.

In addition to the differences in the working memory and the interaction between the semantic and working memories, the high-performance participants could make efficient use of attention resources (stronger lower alpha power [57]) in these complex program constructs (which is the basis of problem solving). In contrast, low-performance participants could not make use of their attention resources efficiently to conduct complex tasks, which influenced their cognitive performance. High-performance participants were able to more effectively develop plans for program comprehension and displayed more top-down processing (internal processing) when achieving their goals (stronger lower alpha power [82]). In contrast, the low-performance participants struggled more with the complex iterations or recursions. They tried to link different parts of the code in order to grasp the program logic [22]. These results are presented in the EEG results, which show differences between the low- and high-performance participants in theta and alpha powers.

B. DIFFERENCES BETWEEN THE HIGH- AND LOW-PERFORMANCE PARTICIPANTS IN COMPREHENDING DIFFICULT PROGRAMMING FUNCTIONS (R1ii, R2)

Our findings imply a strong correlation between brain activity and computer program complexity. In order to explore more completely this correlation, we analyzed and compared learners' EEG responses to difficult programming functions. Here, a function is deemed difficult if more than half of the participants could not correctly understand its meaning (too complex to be processed in their memories), as reflected by what students told us about program complexity in interviews. Consequently, Function #3 of the recursive program was found to be the most difficult (22 incorrectly answering and 20 perceiving a heavy memory load).

Differences in EEG activities are shown in Table 6, which indicates that working-memory ability (theta power) and the interaction between the semantic and working memories (upper alpha power) played key roles in program comprehension for these difficult functions. These results might be explained as follows: understanding difficult functions might also require frequent hypothesis testing which is needed to

TABLE 5. Descriptive statistics and results of Kruskal-Wallis tests of EEG waves for each program construct.

Construct Frequency	Condition (N: High– 17 Low– 16)	Iteration (N: High-18 Low- 15) while loop	 for loop	Recursion (N: High-20 Low- 13) General recursion	 Head-block
theta	High: $\bar{x}$=109379 SD=33128 Low: $\bar{x}$= 82153 SD=18708 p<.036	High: $\bar{x}$=105255 SD= 36901 Low: $\bar{x}$= 82829 SD= 34389 p=.098	High: $\bar{x}$=114189 SD= 39853 Low: $\bar{x}$= 78217 SD= 24553 p<.036	High: $\bar{x}$=118571 SD= 40915 Low: $\bar{x}$= 83788 SD= 35020 p<.036	High: $\bar{x}$=115063 SD= 50909 Low: $\bar{x}$= 85269 SD= 42370 p=.077
lower alpha	High: $\bar{x}$= 27772 SD= 9452 Low: $\bar{x}$=20293 SD= 4222 p<.036	High: $\bar{x}$= 27382 SD= 9577 Low: $\bar{x}$= 21154 SD= 5694 p=.040	High: $\bar{x}$= 27359 SD= 7600 Low: $\bar{x}$= 19454 SD= 4012 p<.036	High: $\bar{x}$= 28427 SD= 9144 Low: $\bar{x}$=20319 SD= 4891 p<.036	High: $\bar{x}$= 28942 SD= 9171 Low: $\bar{x}$= 22105 SD= 8680 p=.047
upper alpha	High: $\bar{x}$= 21713 SD= 7913 Low: $\bar{x}$= 16477 SD= 2843 p<.036	High: $\bar{x}$= 20844 SD= 8628 Low: $\bar{x}$= 17334 SD= 6640 p=.140	High: $\bar{x}$= 22936 SD= 7769 Low: $\bar{x}$= 16724 SD= 2644 p<.036	High: $\bar{x}$=23499 SD= 8528 Low: $\bar{x}$= 16354 SD= 5651 p<.036	High: $\bar{x}$= 24158 SD= 13444 Low: $\bar{x}$= 16135 SD= 6969 p=.141

execute plans involved with program comprehension [82]. Novice learners tended to use the *loop* method to comprehend *recursive functions*, unlike experts, who would use the *stack* method [34], making it even more challenging to track the program logic. Based on our previous research, the high-performance participants tended to organize the code into chunks according to the beacons and formulate their comprehension strategies based on prior knowledge–coupled with their ability to identify problems [22]. For these reasons, the high-performance participants tended to avoid overloading their working memory. They, instead, accessed program chunks and could retrieve prior knowledge from their semantic memories smoothly. The interview results also show that the high-performance participants could recognize the algorithmic patterns (beacons) quickly to grasp the function purposes [31]. A series of functions are shown below which are involved in this process:

A: This function is to check each triple set of numbers and convert into an octal value.

B: $y = y + s[++v]*\text{pow}(2, j\text{-}1)$ is used to convert binary numbers to a decimal form.

Participant A could recognize the instruction chunks as "checking the triples of numbers" and "converting the numbers into octet values". Participant B could recognize the complex instruction $y = y + s[++v]*\text{pow}(2, j\text{-}1)$ as the conversion equation from the binary to decimal form.

Low-performance participants, however, tended to interpret the superficial meanings of the functions based on the instructions–rather than their algorithmic meanings:

C: Indicates some computations and conditional judgements on x.

TABLE 6. The results of Kruskal-Wallis tests of EEG waves for the difficult functions (complex recursion) between the high and low-performance participants.

Frequency	High-performance participants	Low-performance participants	p value
Theta	$\bar{x}$=138180	$\bar{x}$=93924	p<.036
lower alpha	$\bar{x}$=32912	$\bar{x}$=22501	p<.036
upper alpha	$\bar{x}$=28184	$\bar{x}$=18770	p<.036

D: This function allows us to return value c back to the output.

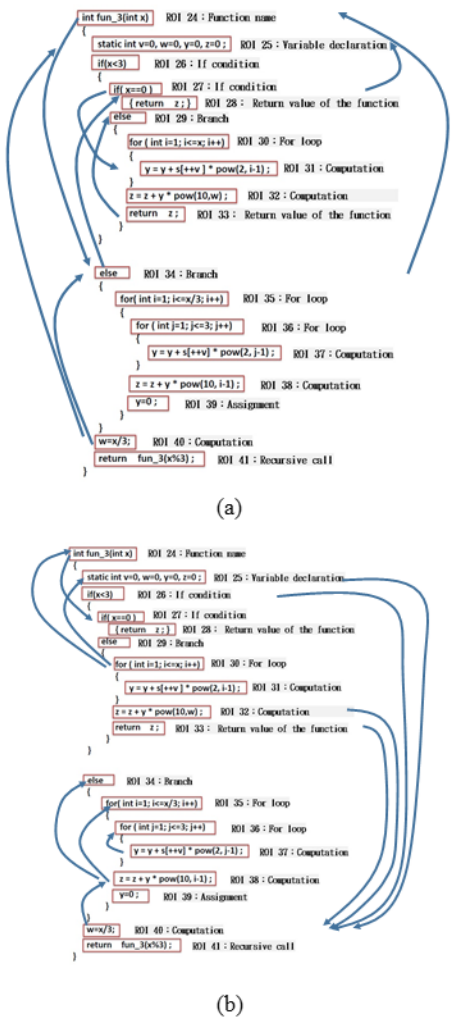
Even though some low-performance participants attempted to find patterns by using simulation, they still performed poorly due to the complex logic, and, it seems, overloading their working memory:

E: This function is used to store the values of $s[]$, like $s[0]*1$, $s[1]*2$, $s[2]*4$, Um…

In order to comprehend more difficult functions, high-performance participants needed better performance in the areas of working and semantic memories. The high-performance participants were, additionally, shown to allocate attention resources more efficiently when attempting to understand more difficult functions (lower alpha power). Our findings with regard to the comprehension of more difficult program functions were similar to the results for general functions discussed in the "*Differences between the high- and low-performance participants for different program constructs*" subsection, which indicates the important roles of working-memory ability, semantic-memory ability, and attention in program comprehension. In short, in the case of difficult functions, hypotheses H1.1, H1.2, and H1.3 were supported by the data. Moreover, all p-values for the three bands are lower than 0.01. Some of the p-values for other functions, however, are not below 0.01, which supports the H1.4 hypothesis.

In order to further understand how the high- and low-performance participants comprehended difficult functions and how EEG activities were influenced by the comprehension process, the ROI sequences derived by eye-tracking were analyzed. The significant ROI sequences that occurred only in either the groups of high-performance participants or low-performance participants are shown and illustrate the specific cognitive processes of high- and low-performance participants (Fig 5).

Based on the eye-tracking result, additional sequences show that the high-performance participants tended to jump between the *if* and *else* blocks (Fig 5(a)). This finding implies that the high-performance participants could recognize and the instructions in the *if-else* blocks and considered the statements in the *if* or *else* blocks as "chunks." They could grasp the whole picture of the program (top-down attention); therefore, they might process program comprehension more effectively without an extra working-memory load [22]. This finding agrees with the EEG results indicating that high-performance participants had a better working-memory ability.



```
int fun_3(int x)                         ROI 24 : Function name
{
  static int v=0, w=0, y=0, z=0 ;        ROI 25 : Variable declaration
  if(x<3)                                ROI 26 : If condition
  {
    if( x==0 )                           ROI 27 : If condition
      {return  z;}                       ROI 28 : Return value of the function
    else                                 ROI 29 : Branch
    for ( int i=1; i<=x; i++)            ROI 30 : For loop
    {
      y = y + s[++v ] * pow(2, i-1) ;    ROI 31 : Computation
    }
    z = z + y * pow(10,w) ;              ROI 32 : Computation
    return   z ;                         ROI 33 : Return value of the function
  }
  }
  else                                   ROI 34 : Branch
  {
    for( int i=1; i<=x/3; i++)           ROI 35 : For loop
    {
      for ( int j=1; j<=3; j++)          ROI 36 : For loop
      {
        y = y + s[++v] * pow(2, j-1) ;   ROI 37 : Computation
      }
      z = z + y * pow(10, i-1) ;         ROI 38 : Computation
      y=0 ;                              ROI 39 : Assignment
    }
  }
  w=x/3;                                 ROI 40 : Computation
  return   fun_3(x%3) ;                  ROI 41 : Recursive call
}
```

FIGURE 5. **The eye-movement results of the complex recursive function for the (a) high- and (b) low-performance participants.**

High-performance participants demonstrated the ability to understand the recursive structure better than low-performers. They were able to track back to the definitions of the recursive function more quickly than low performers. This feature allowed them to by-pass the *arguments* feature once they traced back to the *return* statement (recursive function). High-performance participants were able to simulate the execution process of recursion mentally. In contrast, the low-performance participants tended to stick to the in-block instructions; hence, there were fewer cross-block sequences except for the jump to the end of the program. This finding might be due to the fact that participants tended to struggle with comprehending the program; hence, they jumped to the end of the program frequently from different statements. For example, the right-hand sequences shown in Fig 5(b) were used to were participants endeavored to determine the purpose of the program based on the output variable. This appeared to be an ‘‘opportunistic’’ strategy which did not follow the recursive logic. In addition, the low-performance participants went back-and-forth among *nested for loops* and *conditional statements* fairly randomly (the left-hand sequences in Fig 5(b)), which might indicate that they were ‘‘stumped’’ by the complex logic of *nested for loops* and *conditional statements*. This phenomenon of going back-and-forth undoubtedly increased the complexity of tracing and may have contributed to ‘‘information overload.’’

Listed below are some of the low-performance participants’ responses about the recursive functions as shown from the interviews with participants:

F: I cannot understand how the recursive function executes.

G: I cannot understand the purpose of the statement $z = z + y^*\text{pow}()$ in the nested for loop.

H: This function has too many variables for me to remember.

I: I cannot remember so much information during code tracing without paper and pencil.

One of the low-performance participants even explained that her comprehension as merely *interpreting the program line by line* without *any logic being used*. Additionally, she could not simulate the function execution feature when the interviewer asked her to track the code using a given input.

VII. SUMMARY OF RESEARCH FINDINGS

Our overall results with regard to comprehending basic computer program functions in various types of complex program constructs are summarized as follows. We elucidated three major findings in the study: Firstly, the high-performance participants could recognize beacons. They could, therefore, organize computer code into meaningful chunks based on the relationships among statements. This decreased their burdens upon working-memory (memory loads). Secondly, high-performance participants were more likely to trace the code based on control flow and program logic. They, therefore, displayed high performance regarding the interaction between mental representation in the working memory and programming knowledge in the long-term memory [83]. Our third major finding was as follows: high-performance participants could grasp the global structure of the function calls. They then processed the constructs structurally using top-down strategies based on their plans. In this way, they could employ attention resources more efficiently than the low-performance participants. When utilizing more difficult functions, the results were more obvious.

We can summarize our results as follows. High-performance participants could process program comprehension tasks more efficiently because they could master the logic of programs more quickly based on their knowledge. They could, consequently, structurally separate programs into meaningful chunks in order to conduct their plans. They also were able to optimize limited memory resources in order to process complex program tasks. Low-performance participants, however, were demonstrated to have limited capabilities with regard to compiling programming knowledge and executing their tracings by the production systems [10], [80]. These findings are in agreement with research by Soloway and Ehrlich who indicated that programming experts can apply two types of knowledge effectively: (1) programming plans and (2) the rules of programming discourse [6]. Program plans involve using program fragments effectively to represent action sequences in programming. The rules of programming discourse are used to capture the conventions and govern the composition of the plans [6]. Our findings help to clarify and reinforce the important role of programming knowledge in developing comprehension plans, and also for using the working memory more efficiently. The use of programming plans is vital to develop hypotheses and drive the comprehension process. High- and low-performance participants used the methods of both top-down and bottom-up strategies during comprehension. High-performance participants, however, were shown to use more top-down strategies when tracing computer code which was based on the program logic and their development of detailed plans (stronger lower-alpha wave, also supported by the results of eye tracking and interviews). This is in agreement with claims made by Brooks [10], and Soloway and Ehrlich [6]. The low-performance participants, however, lacked programming knowledge and had a lower working-memory capacity. They were not able to form their plans successfully or simulate their execution mentally. As a result, low-performance participants tended to stick to a *line-by-line sequence* or local instructions, which utilize *bottom-up* approaches.

VIII. THREATS TO VALIDITY

The EEG headset included EEG noise filtering and is comparable to medical-grade systems [72], [73]. The headset, however, in spite of the inclusion of noise filtering, is still prone to producing artifacts with regard to eye blinks and muscle movements [73], which might decrease somewhat the validity of the current research. Additionally, the EEG signals in this study were collected from only one channel at the forehead which records only one aspect brain activity. We hope that the use of a more basic, feasible, and transportable device allows for more ready investigation of learners' cognitive processes in real-world classroom situations. However, this might be another treat to validity. Although previous research has argued that the estimation accuracy using the EEG features obtained from the forehead channel was comparable to that using the EEG features of the whole-head recordings in some applications [84], a follow up investigation using more elaborate medical grade might allow investigators to clarify in a more detailed way the plans, thoughts, and executions of students when engaged in computer programming activities. The recruitment of additional participants from multiple populations of college students might also improve aspects of internal validity. The participants in the current study consisted of a single population of college students studying computer science in East Asia. The sample size was only 33, which might cause a selection bias and affect the results. Participants' years of education in C programming could have affected students' programming comprehension as well. The design of experimental materials and settings was another threat: this study included only two, albeit typical, programs, covering three types of procedural program constructs, and only one programming language (C language). The physical constraints of the distance between the eye tracker and the subject, the position of the eye tracker, might also influence validity. In addition, we prohibited the use of paper and pencil use to force participants to utilize completely their cognitive abilities and focus on the screen, which could possibly have affected their code tracing ability.

When addressing the external validity of the study, we took into account several factors. Since each program had several functions, comprehension might be influenced by the interactions among them. EEG results, however, were derived and based mainly upon individual functions. Future studies could be designed to explore more fully the interactions among functions by linking eye-tracking data more tightly to EEG data. A perceived criticism of the current study might be that mapping between the EEG waves and cognitive processes might not be totally unique [15]—especially with regard to working memory. In addition to working-memory capacity, the theta wave has been associated with other cognitive functions like general problem solving abilities and math comprehension [18], [85]. Most of the functions mentioned in the literature, however, are related to one's working memory. Alpha waves, in particular, have been primarily associated with memory functions. Other cognitive brain activities, however, have been associated with various brain waves and includes processes such as mental effort, intelligence, and brain efficiency [60], [64], [65]. In this study, we focused upon programming-related functions described in previous scientific research that involved programming comprehension (e.g., problem solving, working-memory operation, and semantic understanding). More accurate interpretations may, perhaps, be obtained by integrating external evidence such as measurement of working-memory capacity or interviews dedicated to understanding of subjects' intention and comprehension dynamics.

This study was not intended to reveal the causality among working-memory capacity, programming knowledge, programming strategies, and programming performance but to explore their possible relationships. Based on the integrated results from existing and present research, working-memory capacity programming strategies, and programming knowledge have all been shown to affect performance [3], [6], [86]. Our research, additionally, was able to reveal more details about programming cognition by studying the relationships between working-memory capacity, programming strategies, and performance based on the integrated results from EEGs, eye tracking, and the comprehension test. In order to more effectively determine causality, additional studies and measurements (e.g., working-memory capacity or programming knowledge) may be needed.

IX. CONCLUSION AND SUGGESTIONS

Computer programming is a complicated process requiring knowledge, planning, and the ability to adapt to novel situations. The exploration about why the low-performance learners faced cognitive challenges and difficulties when executing various computer programming tasks is essential for improving programing instruction. This study analyzed EEG activities and eye movements of learners during program comprehension with the aim to further delineate the association between EEG activities and cognition processes used in program comprehension. The inclusions of eye movements along with EEG data was important supporting evidence that helped to validate our results. By comparing the differences in EEG activities between the high- and low-performance participants, we were able to discern possible difficulties learners encountered during the process of program comprehension.

The following conclusions can be drawn from the resulting data. High-performance participants demonstrated a high intensity of theta, lower alpha, and upper alpha brain waves. This is consistent with their higher performance of their working memory, greater attention allocation, and higher interactions between working and semantic memories. This finding implies that the understanding of computer programs involves a high working-memory load and especially with solving complex, difficult programs. Mental computation tasks have shown to be constrained by working-memory capacity [22]. Low-performance participants were shown to have insufficient working memory capacity to solve complex computer programming problems which was supported by the fact that they tended to interpret the superficial meanings of the functions based on the instructions rather than the algorithmic meanings. High-performance participants could, on the other hand, recognize beacons and group program statements into meaningful chunks based algorithmic patterns in their prior-knowledge base. This allowed them to optimize working-memory resources. In order to comprehend the computer program, subjects needed adequate attention resources in order to trace the code. Low-performance participants were shown to have limited attention resources needed to process the tracing tasks. In contrast, the high-performance participants could trace the code globally and structurally to grasp the code logic more effectively–utilizing a limited working memory and attention resources. For understanding and executing the complex program logic, the subjects also needed to transfer the required knowledge from semantic to the working memory. This allowed them to compile their knowledge and map the known patterns to the current code.

Based on these findings, the following pedagogical strategies might be implemented by teachers. Firstly, to enhance students' knowledge with regard to planning, teachers should emphasize the importance of understanding the structure and logic of the program. This should be done prior to learning the syntax or details of codes. Novices can be assisted if they are told that program comprehension is a nonlinear and a dynamic process. Teachers should attempt to help students with understanding program logic first rather the proceeding in a linear fashion with learning syntax. A second pedagogical approach might be to select materials and activities designed to help students in recognizing key beacons within a computer program. Providing supplementary explanations and hints with regard to program logic could be utilized. This could also be couple with visualizations of program chunks. These techniques can assist learners with building and executing their programming plans. This will help students to make more efficient use of working memory and attention resources. The dynamic display of program logic might also help learners simulate the execution process mentally and reduce their cognitive loads. Exercises can, additionally, be designed to help learners practice program segmentation, explanation, and planning.

Our research findings can also be applied to cognitive research in the fields of engineering education, human-computer interaction, artificial intelligence, and neuroscience. The depiction of learners' cognitive processes helps researchers understand more about how humans think and interact with tasks or machines. The data obtained in this study can also be used to help validate and improve the neuroscience instrumentation which may also help to improve instruments and techniques used in biomedical engineering. The use of additional subjects from different demographic groups should be included in future studies in order to obtain a more representative cross-sectional sample. This will improve the nature of data acquired and allow for more broad-reaching, to generalized results. Different program paradigms (e.g., object-oriented programming and visual programming) can also be introduced for exploring additional aspects of programming cognition. Various neuroscience techniques can also be applied in order to analyze students' cognition during learning. This can be done by utilizing different types of instructional media such as web-based learning platforms and video tutorials. Signal processing techniques (e.g., the Wavelet and Fourier Transform) can also be applied in order to transform time-domain EEG powers to time-frequency or frequency-domain signals which will allow investigators to obtain more condensed and efficient features. Clustering and classification algorithms may also be employed to group the participants based on their EEG features to obtain more insights about programming cognition.

REFERENCES

[1] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *Int. J. Comput. Inf. Sci.*, vol. 8, no. 3, pp. 219–238, Jun. 1979.

[2] J. J. Cañas, M. T. Bajo, and P. Gonzalvo, "Mental models and computer programming," *Int. J. Hum.-Comput. Stud.*, vol. 40, no. 5, pp. 795–811, May 1994.

[3] G. R. Bergersen and J.-E. Gustafsson, "Programming skill, knowledge, and working memory among professional software developers from an investment theory perspective," *J. Individual Differences*, vol. 32, no. 4, pp. 201–209, Jan. 2011.

[4] R. Bednarik, "Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations," *Int. J. Hum.-Comput. Stud.*, vol. 70, no. 2, pp. 143–155, Feb. 2012.

[5] N. Pennington, "Stimulus structures and mental representations in expert comprehension of computer programs," *Cognit. Psychol.*, vol. 19, no. 3, pp. 295–341, Jul. 1987.

[6] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Trans. Softw. Eng.*, vol. SE-10, no. 5, pp. 595–609, Sep. 1984.

[7] A. F. Blackwell, M. Petre, and L. Church, "Fifty years of the psychology of programming," *Int. J. Hum.-Comput. Stud.*, vol. 131, pp. 52–63, Nov. 2019.

[8] I. Crk and T. Kluthe, "Toward using alpha and theta brain waves to quantify programmer expertise," in *Proc. 36th Annu. Int. Conf. Eng. Med. Biol. Soc.*, Aug. 2014, pp. 5373–5376.

[9] S. Lee, D. Hooshyar, H. Ji, K. Nam, and H. Lim, "Mining biometric data to predict programmer expertise and task difficulty," *Cluster Comput.*, vol. 21, no. 1, pp. 1097–1107, Mar. 2018.

[10] R. Brooks, "Towards a theory of the comprehension of computer programs," *Int. J. Man-Mach. Stud.*, vol. 18, no. 6, pp. 543–554, Jun. 1983.

[11] Y. G. Guéhéneuc, "A theory of program comprehension: Joining vision science and program comprehension," *Int. J. Soft. Sci. Comput. Intell.*, vol. 1, no. 2, pp. 54–72, 2009.

[12] V. J. Shute, "Who is likely to acquire programming skills?" *J. Educ. Comput. Res.*, vol. 7, no. 1, pp. 1–24, Feb. 1991.

[13] P. Sauseng, B. Griesmayr, R. Freunberger, and W. Klimesch, "Control mechanisms in working memory: A possible function of EEG theta oscillations," *Neurosci. Biobehav. Rev.*, vol. 34, no. 7, pp. 1015–1022, Jun. 2010.

[14] B. Floyd, T. Santander, and W. Weimer, "Decoding the representation of code in the brain: An fMRI study of code review and expertise," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. (ICSE)*, May 2017, pp. 175–186.

[15] M. X. Cohen, "Where does EEG come from and what does it mean?" *Trends Neurosci.*, vol. 40, no. 4, pp. 208–218, Apr. 2017.

[16] J. Katona, "Clean and dirty code comprehension by eye-tracking based evaluation using GP3 eye tracker," *Acta Polytechnica Hungarica*, vol. 18, no. 1, pp. 79–99, 2021.

[17] T. V. Gog, L. Kester, F. Nievelstein, B. Giesbers, and F. Paas, "Uncovering cognitive processes: Different techniques that can contribute to cognitive load research and instruction," *Comput. Hum. Behav.*, vol. 25, no. 2, pp. 325–331, Mar. 2009.

[18] T. Hinault and P. Lemaire, "What does EEG tell us about arithmetic strategies? A review," *Int. J. Psychophysiol.*, vol. 106, pp. 115–126, Aug. 2016.

[19] A. Gevins, M. E. Smith, L. McEvoy, and D. Yu, "High-resolution EEG mapping of cortical activation related to working memory: Effects of task difficulty, type of processing, and practice," *Cerebral Cortex*, vol. 7, no. 4, pp. 374–385, Jun. 1997.

[20] Y. G. Pavlov and B. Kotchoubey, "EEG correlates of working memory performance in females," *BMC Neurosci.*, vol. 18, no. 1, pp. 1–14, Dec. 2017.

[21] W. Klimesch, "EEG alpha and theta oscillations reflect cognitive and memory performance: A review and analysis," *Brain Res. Rev.*, vol. 29, nos. 2–3, pp. 169–195, Apr. 1999.

[22] Y.-T. Lin, C.-C. Wu, T.-Y. Hou, Y.-C. Lin, F.-Y. Yang, and C.-H. Chang, "Tracking students' cognitive processes during program debugging—An eye-movement approach," *IEEE Trans. Educ.*, vol. 59, no. 3, pp. 175–186, Aug. 2016.

[23] S. Wiedenbeck and V. Ramalingam, "Novice comprehension of small programs written in the procedural and object-oriented styles," *Int. J. Human-Comput. Stud.*, vol. 51, no. 1, pp. 71–87, Jul. 1999.

[24] F. Détienne, *Software Design-Cognitive Aspect*. London, U.K.: Springer, 2001.

[25] L. Hohmann, *Journey of the Software Professional: The Sociology of Software Development*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.

[26] S. Letovsky, "Cognitive processes in program comprehension," *J. Syst. Softw.*, vol. 7, no. 4, pp. 325–339, Dec. 1987.

[27] C. Schulte, T. Clear, A. Taherkhani, T. Busjahn, and J. H. Paterson, "An introduction to program comprehension for computer science educators," in *Proc. ITiCSE Work. Group Rep. Work. Group*, 2010, pp. 65–86.

[28] J. M. Burkhardt, F. Détienne, and S. Wiedenbeck, "Mental representations constructed by experts and novices in object-oriented program comprehension," in *Human-Computer Interaction*. Boston, MA, USA: Springer, 1997.

[29] V. Fix, S. Wiedenbeck, and J. Scholtz, "Mental representations of programs by novices and experts," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst.*, 1993, pp. 74–79.

[30] C. L. Corritore and S. Wiedenbeck, "What do novices learn during program comprehension?" *Int. J. Hum.-Comput. Interact.*, vol. 3, no. 2, pp. 199–222, Jan. 1991.

[31] M. E. Crosby, J. Scholtz, and S. Wiedenbeck, "The roles beacons play in comprehension for novice and expert programmers," in *Proc. 14th Workshop Psychol. Program. Interest Group*, 2002, pp. 58–73.

[32] C. Aschwanden and M. Crosby, "Code scanning patterns in program comprehension," in *Proc. 39th Hawaii Int. Conf. Syst. Sci.*, 2006, pp. 1–10.

[33] A. Taherkhani, (2013). *Automatic Algorithm Recognition Based on Programming Schemas and Beacons*. [Online]. Available: http://lib.tkk.fi/Diss/2013/isbn9789526049908/isbn9789526049908.pdf

[34] D. Dicheva and J. Close, "Mental models of recursion," *J. Educ. Comput. Res.*, vol. 14, no. 1, pp. 1–23, Jan. 1996.

[35] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proc. 36th Int. Conf. Softw. Eng.*, New York, NY, USA, May 2014, pp. 378–389.

[36] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann, "Measuring neural efficiency of program comprehension," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, New York, NY, USA, 2017, pp. 140–150.

[37] J. Duraes, H. Madeira, J. Castelhano, C. Duarte, and M. C. Branco, "WAP: Understanding the brain at software debugging," in *Proc. Int. Symp. Softw. Reliab. Eng. (ISSRE)*, 2016, pp. 87–92.

[38] M. V. Kosti, K. Georgiadis, D. A. Adamos, N. Laskaris, D. Spinellis, and L. Angelis, "Towards an affordable brain computer interface for the assessment of programmers' mental workload," *Int. J. Hum.-Comput. Stud.*, vol. 115, pp. 52–66, Jul. 2018.

[39] T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, and D. M. German, "Quantifying programmers' mental workload during program comprehension based on cerebral blood flow measurement: A controlled experiment," in *Proc. 36th Int. Conf. Softw. Eng.*, May 2014, pp. 448–451.

[40] Y. Ikutani and H. Uwano, "Brain activity measurement during program comprehension with NIRS," *Int. J. Netw. Distrib. Comput.*, vol. 2, no. 4, pp. 259–268, 2014.

[41] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *Proc. 26th Conf. Program Comprehension*, May 2018, pp. 286–296.

[42] W. Klimesch, "Alpha-band oscillations, attention, and controlled access to stored information," *Trends Cogn. Sci.*, vol. 16, no. 12, pp. 606–617, Dec. 2012.

[43] L. M. Ward, "Synchronous neural oscillations and cognitive processes," *Trends Cognit. Sci.*, vol. 7, no. 12, pp. 553–559, Dec. 2003.

[44] Z. Huang, A. Javaid, V. K. Devabhaktuni, Y. Li, and X. Yang, "Development of cognitive training program with EEG headset," *IEEE Access*, vol. 7, pp. 126191–126200, 2019.

[45] P. Sauseng, W. Klimesch, M. Schabus, and M. Doppelmayr, "Fronto-parietal EEG coherence in theta and upper alpha reflect central executive functions of working memory," *Int. J. Psychophysiol.*, vol. 57, no. 2, pp. 97–103, 2005.

[46] M. J. Kahana, D. Seelig, and J. R. Madsen, "Theta returns," *Current Opinion Neurobiol.*, vol. 11, no. 6, pp. 739–744, Dec. 2001.

[47] P. Sauseng and W. Klimesch, "What does phase information of oscillatory brain activity tell us about cognitive processes?" *Neurosc. Biobehav. Rev.*, vol. 32, no. 5, pp. 1001–1013, Jul. 2008.

[48] W. Klimesch, M. Doppelmayr, H. Russegger, and T. Pachinger, "Theta band power in the human scalp EEG and the encoding of new information," *Neuroreport*, vol. 7, no. 7, pp. 1235–1240, 1996.

[49] M. J. Kahana, R. Sekuler, J. B. Caplan, M. Kirschen, and J. R. Madsen, "Human theta oscillations exhibit task dependence during virtual maze navigation," *Nature*, vol. 399, no. 6738, pp. 781–784, 1999.

[50] J. B. Caplan, J. R. Madsen, S. Raghavachari, and M. J. Kahana, "Distinct patterns of brain oscillations underlie two basic parameters of human maze learning," *J. Neurophysiol.*, vol. 86, no. 1, pp. 368–380, Jul. 2001.

[51] H.-C. She, T.-P. Jung, W.-C. Chou, L.-Y. Huang, C.-Y. Wang, and G.-Y. Lin, "EEG dynamics reflect the distinct cognitive process of optic problem solving," *PLoS ONE*, vol. 7, no. 7, Jul. 2012, Art. no. e40731.

[52] B. De Smedt, R. H. Grabner, and B. Studer, "Oscillatory EEG correlates of arithmetic strategy use in addition and subtraction," *Exp. Brain Res.*, vol. 195, no. 4, pp. 635–642, Jun. 2009.

[53] S. Sandkähler and J. Bhattacharya, "Deconstructing insight: EEG correlates of insightful problem solving," *PLoS ONE*, vol. 3, no. 1, Jan. 2008, Art. no. e1459.

[54] O. Jensen and C. D. Tesche, "Frontal theta activity in humans increases with memory load in a working memory task," *Eur. J. Neurosci.*, vol. 15, no. 8, pp. 1395–1399, Apr. 2002.

[55] W. Klimesch, B. Schack, and P. Sauseng, "The functional significance of theta and upper alpha oscillations," *Exp. Psychol.*, vol. 52, no. 2, pp. 99–108, Sep. 2006.

[56] D. Jokisch and O. Jensen, "Modulation of gamma and alpha activity during a working memory task engaging the dorsal or ventral stream," *J. Neurosci.*, vol. 27, no. 12, pp. 3244–3251, Mar. 2007.

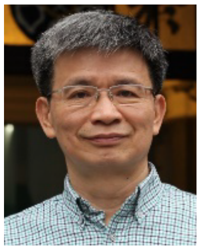
[57] W. Klimesch, H. Schimke, and G. Pfurtscheller, "Alpha frequency, cognitive load and memory performance," *Brain Topography*, vol. 5, no. 3, pp. 241–251, Mar. 1993.

[58] W. Klimesch, "EEG-alpha rhythms and memory processes," *Int. J. Psychophysiol.*, vol. 26, nos. 1–3, pp. 319–340, Jun. 1997.
[59] W. Klimesch, M. Doppelmayr, T. Pachinger, and B. Ripper, "Brain oscillations and human memory: EEG correlates in the upper alpha and theta band," *Neurosci. Lett.*, vol. 238, nos. 1–2, pp. 9–12, Nov. 1997.
[60] N. Jaušovec, "Differences in cognitive processes between gifted, intelligent, creative, and average individuals while solving complex problems: An EEG study," *Intelligence*, vol. 28, no. 3, pp. 213–237, Sep. 2000.
[61] M. Benedek, S. Bergner, T. Könen, A. Fink, and A. C. Neubauer, "EEG alpha synchronization is related to top-down processing in convergent and divergent thinking," *Neuropsychologia*, vol. 49, no. 12, pp. 3505–3511, Oct. 2011.
[62] P. Sauseng, W. Klimesch, M. Doppelmayr, T. Pecherstorfer, R. Freunberger, and S. Hanslmayr, "EEG alpha synchronization and functional coupling during top-down processing in a working memory task," *Hum. Brain Mapping*, vol. 26, no. 2, pp. 148–155, 2005.
[63] B.-K. Min and H.-J. Park, "Task-related modulation of anterior theta and posterior alpha EEG reflects top-down preparation," *BMC Neurosci.*, vol. 11, no. 1, Dec. 2010, Art. no. 79.
[64] N. Jaušovec, "Differences in EEG alpha activity related to giftedness," *Intelligence*, vol. 23, no. 3, pp. 159–173, Nov. 1996.
[65] R. J. Haier, B. Siegel, C. Tang, L. Abel, and M. S. Buchsbaum, "Intelligence and changes in regional cerebral glucose metabolic rate following learning," *Intelligence*, vol. 16, nos. 3–4, pp. 415–426, Jul. 1992.
[66] G. Pfurtscheller, "Induced oscillations in the alpha band: Functional meaning," *Epilepsia*, vol. 44, no. s12, pp. 2–8, Dec. 2003.
[67] W. Klimesch, P. Sauseng, and S. Hanslmayr, "EEG alpha oscillations: The inhibition-timing hypothesis," *Brain Res. Rev.*, vol. 53, no. 1, pp. 63–88, 2007.
[68] J. Katona and A. Kovari, "Examining the learning efficiency by a brain-computer interface system," *Acta Polytechnica Hungarica*, vol. 15, no. 3, pp. 251–280, 2018.
[69] W. K. Y. So, S. W. H. Wong, J. N. Mak, and R. H. M. Chan, "An evaluation of mental workload with frontal EEG," *PLoS ONE*, vol. 12, no. 4, Apr. 2017, Art. no. e0174949.
[70] G. U. Navalyal and R. D. Gavas, "A dynamic attention assessment and enhancement tool using computer graphics," *Hum.-Centric Comput. Inf. Sci.*, vol. 4, no. 1, p. 11, Dec. 2014.
[71] A. R. Elshenaway and S. K. Guirguis, "Adaptive thresholds of EEG brain signals for IoT devices authentication," *IEEE Access*, vol. 9, pp. 100294–100307, 2021.
[72] S. J. Johnstone, R. Blackman, and J. M. Bruggemann, "EEG from a single-channel dry-sensor recording device," *Clin. EEG Neurosci.*, vol. 43, no. 2, pp. 112–120, Apr. 2012.
[73] E. Ratti, S. Waninger, C. Berka, G. Ruffini, and A. Verma, "Comparison of medical and consumer wireless EEG systems for use in clinical trials," *Frontiers Hum. Neurosci.*, vol. 11, p. 398, Aug. 2017.
[74] J. Katona, "Examination and comparison of the EEG based attention test with CPT and TOVA," in *Proc. IEEE 15th Int. Symp. Comput. Intell. Informat. (CINTI)*, Nov. 2014, pp. 117–120.
[75] B. T. Carter and S. G. Luke, "Best practices in eye tracking research," *Int. J. Psychophysiol.*, vol. 155, pp. 49–62, Sep. 2020.
[76] Z. Sharafi, B. Sharif, Y. G. Guéhéneuc, A. Begel, R. Bednarik, and M. Crosby, "A practical guide on conducting eye tracking studies in software engineering," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3128–3174, 2020.
[77] R. Bakeman and J. M. Gottman, *Observing Interaction: An Introduction to Sequential Analysis*, 1st ed. New York, NY, USA: Cambridge Univ. Press, 1986.
[78] D. W. Nordstokke and B. D. Zumbo, "A new nonparametric Levene test for equal variances," *Psicológica*, vol. 31, no. 2, pp. 401–430, 2010.
[79] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Stat. Soc., B Methodol.*, vol. 57, pp. 289–300, Jan. 1995.
[80] F. Smith and D. L. Holmes, "The independence of letter, word, and meaning identification in reading," *Reading Res. Quart.*, vol. 1, pp. 394–415, Apr. 1971.
[81] K. F. Wender, S. Franz, and B. Heinz-Dieter, *Cognition and Computer Programming*. Bristol, U.K.: Intellect Books, 1995.
[82] S. Wiedenbeck, "Beacons in computer program comprehension," *Int. J. Man-Mach. Stud.*, vol. 25, no. 6, pp. 697–709, Dec. 1986.
[83] N. Mackintosh, "The fractionation of working memory maps onto different components of intelligence," *Intelligence*, vol. 31, no. 6, pp. 519–531, Dec. 2003.
[84] S. Myllymaa, A. Muraja-Murro, S. Westeren-Punnonen, T. Hukkanen, R. Lappalainen, E. Mervaala, J. Töyräs, K. Sipilä, and K. Myllymaa, "Assessment of the suitability of using a forehead EEG electrode set and chin EMG electrodes for sleep staging in polysomnography," *J. Sleep Res.*, vol. 25, no. 6, pp. 636–645, Dec. 2016.
[85] J. B. B. Earle, P. Garcia-Dergay, A. Manniello, and C. Dowd, "Mathematical cognitive style and arithmetic sign comprehension: A study of EEG alpha and theta activity," *Int. J. Psychophysiol.*, vol. 21, no. 1, pp. 1–13, Jan. 1996.
[86] K. B. McKeithen, J. S. Reitman, H. H. Rueter, and S. C. Hirtle, "Knowledge organization and skill differences in computer programmers," *Cognit. Psychol.*, vol. 13, no. 3, pp. 307–325, Jul. 1981.

YU-TZU LIN (Member, IEEE) received the B.S. and M.S. degrees in information and computer education from National Taiwan Normal University, and the Ph.D. degree in computer science from National Taiwan University. She is currently an Associate Professor with the Graduate Institute of Information and Computer Education, National Taiwan Normal University. Her research interests include computer science education, educational technologies, social network analysis, digital content analysis, multimedia security, pattern recognition, and image processing.

YI-ZHI LIAO received the B.S. degree in industrial education from the National Changhua University of Education, Taiwan, and the M.S. degree in computer science education from National Taiwan Normal University, in 2014. He is currently a Computer Science Teacher with New Taipei San-Chung Commercial and Industrial Vocational High School. His research interests include computer science education, programming instruction, and neuroscience.

XIAO HU received the master's degree in computer science and the Ph.D. degree in library and information science from the University of Illinois. She is currently an Associate Professor with the Faculty of Education, The University of Hong Kong. Her research interests include learning analytics, data science, and cultural informatics.



CHENG-CHIH WU received the B.Ed. and M.Ed. degrees in industrial education from National Taiwan Normal University, and the Ph.D. degree in computer science education from The University of Texas at Austin. He was the Chairman of the Department of Information and Computer Education (now, a Graduate Institute), the Director of the Information Technology Center, and the Vice President of the Academic Affairs with National Taiwan Normal University. He is currently a Distinguished Professor with the Graduate Institute of Information and Computer Education, National Taiwan Normal University. He is also a President of the National Taiwan Normal University. He has been playing a key role in Taiwan's ICT/CS education, including leading the task forces for developing national ICT master plan and revising national K-12 computing curriculum, and serving as the Principal Investigator for Intel Education programs in Taiwan. His research interests include K-12 computing curriculum, e-learning, and ICT-integrated learning.