Neurophysiological Measure of Effort in Program Comprehension

by Brajesh Karna, Bachelor of Science

A Thesis Submitted in Partial
Fulfillment of the Requirements
for the Degree of
Master of Science
in the field of Computer Science

Advisory Committee:

Igor Crk, PhD., Chair

Dennis Bouvier, PhD.

Mark McKenney, PhD.

Graduate School
Southern Illinois University Edwardsville
December, 2017

ABSTRACT

NEUROPHYSIOLOGICAL MEASURE OF EFFORT IN PROGRAM COMPREHENSION

by

BRAJESH KARNA

Chairperson: Igor Crk, PhD.

The recent proliferation of inexpensive electroencephalography (EEG) devices is fueling a rising interest in associating detectable indicators of brain activity with human performance factors. In this thesis, the focus is on programmer effort in program comprehension tasks. Traditionally, measures of effort are made using self-reported surveys (NASA-TLX), task timing, and task accuracy. This work explores the feasibility of using EEG to produce a more direct and quantitative measure of effort. Effort is measured across a number of tasks with varying difficulty and comparisons are made between traditional and EEG measures of effort. Initially, the program comprehension tasks are ranked in order of complexity as computed by a number of classic software complexity metrics, such as Halstead's complexity metrics and McCabe's cyclomatic complexity. Likewise, we compute a ranking of tasks based on observed effort as a basis of comparison between EEG and complexity measures.

ACKNOWLEDGEMENTS

I wish to express my enormous gratitude to my supervisor and my thesis advisor Dr. Igor Crk of the Southern Illinois University at Edwardsville for his continuous support during my Master's study and in this research work. I would like to thank for his patience, motivation, immense knowledge and his guidance throughout this research work.

I would also like to thank Mr. Cagatay Bilgin from Industrial engineering through URCA program for helping me with conducting the experiment and also all the participants for their participation in the experiment.

I would also like to acknowledge Dr. Dennis J Bouvier and Dr. Mark McKenney of the SIU at Edwardsville as being on my thesis committee and the second reader of this thesis and I am grateful for their very valuable comments on this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

Brain-computer interfaces (BCI) based on electroencephalography (EEG) are widely used in research for monitoring electric signals generated by the brain. This approach is relatively new in the area of program comprehension. Building on prior work, which focused on measures of expertise, this work likewise uses EEG-based, passive BCI measurements to determine programmer effort in program comprehension tasks. Where prior work fixed the difficulty of tasks and used a sample population with a range of expertise, this work fixes the sample population to a narrow range of expertise, varying the tasks instead.

Software complexity measures, like Halstead's measures of programming effort based on counting operands and operators or McCabe's cyclomatic complexity based on counting the number of linearly independent paths through a program, provide an approximation of effort required to write or understand code. These same measures are used here to ensure that, at least in approximation, the programming tasks represent a range of complexity, and so intuitively, a range of difficulty which ought to elicit different levels of effort. Using EEG, this study explores whether near-direct measurements of effort reflect the expected effort, given by the classic software complexity measures.

Additionally, participants were asked to report on their perceived effort on each task after the experiments. This was done with a modified NASA-TLX pencil-and-paper instrument.

Throughout the rest of the paper, we explore these findings, identifying brain waves that appears in problem solving. This study may help software development agencies, institutions or future authors to set a guideline about the effort for a particular algorithm to be understood or written. Thus, this work may help better understand the code complexity with respect to programmer effort.

This paper proceeds as follows: Chapter 2 will discuss related work, specifically the EEG, cognitive load, traditional complexity metric, and program comprehension; Chapter 3 focuses on procedure of the experiment, experimental tasks and signal analysis; Chapter 4 and 5 provides details about the findings and statistical analysis; Chapter 6 and 7 will discuss about future works and conclude.

CHAPTER 2

Background

This chapter provides an overview of the past work relating to EEG, program complexity measures, and the role of working memory in program comprehension. This chapter explains the following: Brief introduction to EEG (2.1), Alpha and Theta waves (2.2), theories of cognitive load (2.3), and software complexity metrics (2.4).

2.1 EEG

Neurons are a type of cell which transmit information through electrical signal in the brain . After the development of electroencephalography in 1924 [Ber29] it has been used for detecting and recording the electrical impulses produced by the sum of the synchronized activity of neurons. Electrodes are placed on scalp and the detected signal are then amplified by a factor of approximately 1000 to record them. With the help of EEG now it is possible to view the neural activity in the form of electrical potential generated in the brain ranges from $5\mu V$ to $100\mu V$. Berger et. al. [Ber29] provided evidence that human brain has a certain rhythm. He found that when at rest with closing eyes the electrical activity in brain produces high amplitude oscillation (10Hz) and he termed as "alpha rhythm" and with open eyes faster rhythmic activity was observed with lower amplitude and the 10Hz "alpha rhythm" was dominated an so he called it "beta rhythm". Those synchronized activity is broadly categorized as various rhythmic waves.

By the help of EEG broad range of signals having different frequency, amplitude and phase are captured and recorded. Distinct oscillatory activity has been detected within those frequency bands. Alpha and theta waves are specifically of interest in this study, as these are related to cognitive load [Kli99]. The alpha wave falls in the frequency range of 8Hz to 13Hz, theta falls within the range of 4Hz to 8Hz. The alpha frequency is related to cognitive performance. The theta wave is seen as complementary to alpha: where the

amplitude of one increases with performance, the other is generally expected to decrease and vice versa [Kli99], Delta wave falls within the range of 1Hz to 4Hz and are normally present during deep sleep [HPS02], beta falls within the range of 13Hz to 30Hz and is widely considered to reflect motor activity [Bak07]. Gamma are withing the frequency range of 30Hz to 70Hz and are confounded by the presence of ambient electrical noise. Beta and gamma waves are not of interest for this study.

There are other modern neuroimaging tools used in the market such as functional magnetic resonance imaging (fMRI), positron emission tomography (PET), but they rely on hemodynamic response which means observing the localized blood volume and oxygenation changes in the brain but EEG provides excellent temporal resolution in the range of milliseconds and records neural activity in the real time [SK08], electrocorticogram which requires surgical procedure to install into the skull, and magnetoencephalogram which is very expensive so is commonly not used. This makes EEG the equipment of choice to use broadly in research.

2.2 Alpha and Theta Waves

Alpha brain waves are most prominently observed under deep relaxation when the eyes are closed with no complex cognitive tasks first reported in [Ber29]. These waves, arising from synchronous and coherent electrical activity of thalamic pacemaker cells in brain, may be detected by either electroencephalography (EEG) or magnetoencephalography (MEG). Klimesch, et. al, reported that the peak frequency of alpha waves falls within the frequency range of 8Hz to 13Hz in healthy adult person and the mean frequency of alpha power can be calculated with the Equation 2.1 [Kli99]. While the functional significance of alpha brain wave is unknown, they are generally seen as most prominent during cognitive inactivity [Kli96]. In our study it was observed that when the brain is relaxing with the absence of a task the alpha power increases, confirming prior observations [KSH07, Kli99, CBCG06, JGKL02] and the decrease in alpha power is detected at the presence of a task

which shows there is cognitive load and occurs when group of neurons activate to fulfill the task demand [KSH07, RMT07]. Prior studies [Ber29, Kli99] have also shown that alpha slightly desynchronizes with open eyes so to find the basis for computing relative measure for the trial and rest Fourier transform based power estimation was used.

$$\frac{\sum(a(f) \times f))}{(\sum a(f))} \tag{2.1}$$

where power spectral estimates at frequency f are denoted by $a(f)$

Alpha frequency varies between individual and also with the age group. Is has been shown the increase of alpha frequency from childhood to adulthood and decrease with increasing age [Kli99] which can be seen from the equation 2.2.

$$Peak\ Alpha\ Frequency\ = 11.95 - 0.053 * Age \tag{2.2}$$

While analyzing the signal in this study, rather than computing the overall alpha frequency, we computed the Individual Alpha Frequency (IAF) for each participant. IAF is used to define the alpha frequency range as three sub-bands [Kli99], termed the Lower 1 Alpha (L1A), which ranges from IAF-4 to IAF-2, Lower 2 Alpha (L2A), ranging from IAF-2 to IAF, and Upper Alpha (UA), ranging from IAF to IAF+2. We also included theta, which falls in the range of IAF-6 to IAF-4. Generally, power at the theta frequencies is seen to vary inversely with the power at the alpha frequency.

Event-related desynchronization (ERD) is when all the neurons do not synchronize together and this phenomenon can be seen when there is any observed task demand. ERD can be calculated as the percentage of band power change between the resting period preceding a trial and the trial itself.

$$ERD = \frac{(band\ power)_r - (band\ power)_t)}{(band\ power)_r} \times 100 \tag{2.3}$$

$$Where\ r\ =\ rest\ and\ t\ = trial$$

The duration of the ERD and task duration are closely related [KSSW90], meaning that we can use the entire task period for evaluating the relative change in alpha power with respect to the resting period. Briefly then, when more desynchronization is observed in the L1A sub-band the attentional demands of the given task are higher for the given participant. When more desynchronization is observed in the UA sub-bands, it is indicative of more semantic memory processing [Kli99]. These two sub-bands are therefore the best candidates for attaining a workload estimate as it relates to participant expertise, including something as specific as computer programming expertise.

2.3 Cognitive Load

The Cognitive Load Theory (CLT) which shows how the working memory and long term memory are connected in human. The Working Memory, has two main characteristics, First, it is very limited in capacity [Mil56, Cow01]. Miller et. al. shows the capacity to hold item in working memory is 7 but based on recent study Cowan et. al. shows the number to be 4. Second, duration is very limited [PP59]. Peterson et. al. shows the working memory can only hold information for few seconds and it takes 20 seconds for information to be completely lost. Working memory is the one which is responsible for information retrieval, processing, and integration [Bad92]. Spectral change in alpha and theta relates to the performance of working memory [Kli99, SAK11]. It can be believed that the phasic change in alpha and theta observed during the experiment while participant solves the programming tasks shows the working memory load. With simple tasks like single assignment variable task the desynchronization is low and with more complex task where the working memory has to perform more computation the desynchronization is observed high.

In the past CLT research were performed based on the self reported measures for participant rating their cognitive effort on an ordinal scale immediately after the tasks [Paa92, Har06]. The working memory performance were calculated based on the

accuracy and time to perform tasks and based on the participant self reported data which are approximate. But, by the help of EEG by observing the phasic change in alpha and theta the real time performance of cognitive load can be reported.

2.4 Software Complexity Metrics

Software complexity is an interesting area of research because it can be used for projection of cost, allocation of manpower, evaluation of program and also programmers. The complexity are based on number of factors such as size of the program, number of control structures, number of operators and operators in the program etc. To fully understand the concept of software complexity metric we should know about what makes the program complex. For example, computer complexity can be determined by computation capacity, storage requirement and the execution time, similarly for programmer, complexity can be determined by testing, modifying, coding the software. Halstead et. al. explain the term "software complexity" as the interaction between a program and a programmer working on some task [Hal77].

In this study, software complexity metrics are used to approximate the expected effort needed to comprehend the task, which is a snippet of code with some expected output. Tasks are ordered by difficulty, i.e. expected effort, based on one or more software complexity metrics, to make comparison with EEG-based measures more direct.

2.4.1 Lines of code (LOC)

Counting lines of code (LOC or SLOC) is a well known, simple metric for measuring the size of a software product, typically factoring into sizing and estimation of cost for software products [BAC00]. A significant number of commercially available estimation tools (such as COCOMO II, True-S, SLIM, and CostExpert), use the LOC metric and tend to produce different results [NDRTB07], stemming from varying interpretations of physical and logical line counts. Clearly, there is a need for a consistent and objective

measure of complexity.

Physical and Logical lines of code are the two most popularly accepted source lines of code (SLOC) measures. Counting with physical source lines of code (PSLOC) is performed by counting all the lines excluding white spaces, blanks and comments and are independent of any languages. On the other hand logical source lines of code (LSLOC) are based on counting statements which basically are terminated by semicolon. If the logical statements are span into many lines or it is in one line is counted once. Some higher languages not having semicolon in their statements contradicts the logical source lines of code [NDRTB07].

Physical and logical source lines of code counts based on Nguyen's SLOC counting standard [NDRTB07] are performed on all experimental tasks and are included in Table 4.1.

Table 2.1 summarizes the rules for counting PSLOC and LSLOC. Table 2.2 summarizes counting rules for higher-order languages, like C, C++, Java, and C#. Additionally, two examples are given: Example in section 2.4.2 shows the LOC calculations and results for a simple task used in the study, while Example in section 2.4.3 shows the counts for a more complex task.

Measurement Unit	**PSLOC (count per line)**	**LSLOC (count per line)**
Lines which execute		
statement	1	language independent
Lines which do not execute		
declaration	1	1 per declaration
compiler directives	1	for each directive

Table 2.1: Physical and logical SLOC counting rule

Measurement Unit	**LSLOC (count per occurrences)**
conditional statements	1
loops	1
jumps	1
expressions	1
general statements which ends with semicolon	1
braces, block delimeters	1 per pair
directives	1
declarations eg. struct class etc.	1

Table 2.2: Logical SLOC rules for counting in higher level languages

2.4.2 Example 1: lines of code

```
public class task2
{
   public static void main(String arg[])
   {
      int a = 10;
      int b = 20;
      a = b;
      System.out.format("%d %d", a, b);
   }
}
```

Physical SLOC	**Logical SLOC**
10	6

Table 2.3: Source lines of code example 1

2.4.3 Example 2: lines of code

```
public class task23
{
```

```
   private static int foo(int a)
   {
      return a*2;
   }
   public static void main(String arg[])
   {
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 3; i++)
      {
         array[i] = foo(array[i]);
      }
      System.out.format("%d %d %d", array[0], array[1], array[2]);
   }
}
```

Physical SLOC	**Logical SLOC**
17	9

Table 2.4: Source lines of code example 2

2.4.4 Halstead's complexity metric (HCM)

HCM is a widely used software metric, named after Maurice Halstead, who, in 1977, attempted to formalize software complexity analysis on the basis of counting operators and operands. HCM is a model used for predicting the time and effort required for the writing of software of varying complexity. In short, Halstead views a program as a collection of operators and their associated operands and so the basis of HCM is the counting of the numbers of operators and operands in the program. The main objective is to measure program length, volume, difficulty, effort, and time required to write the program [Hal77]. Based on the Halstead metric any symbols which are used for representing data are considered operands and any symbols or keywords of the language which specify an algorithmic action are operators. Halstead himself defines operands as any variable or constant, and anything that is not an operand is either an operator or part of one.

The base measures can be collected from:

$$n_1 = number\ of\ unique\ operator$$

$$n_2 = number\ of\ unique\ operands$$

$$N_1 = total\ number\ of\ operators$$

$$N_2 = total\ number\ of\ operands$$

$$Program\ Vocabulary\ \ (n) = n_1 + n_2 \tag{2.4}$$

$$Program\ Length\quad (N) = N_1 + N_2 \tag{2.5}$$

$$Calculated\ program\ length\quad (H) = n_1 * \log_2 n_1 + n_2 \log_2 n_2 \tag{2.6}$$

$$Volume\quad (V) = N \times log_2 n\ bits \tag{2.7}$$

$$Difficulty\quad (D) = \frac{n_1}{2} \times \frac{N-2}{n_2} \tag{2.8}$$

$$Effort\quad (E) = D \times V \tag{2.9}$$

The time to solve the program is calculate by equation:

$$Time\ required\ to\ program\quad (T) = \frac{E}{18}\ seconds \tag{2.10}$$

HCM does not take into account the logical structure of the code, but the computation is easy and are independent of programming languages. Halstead also provides an equation to predict effort, time required to solve the problem and the bug density [Hal77].

Halstead complexity measures for all experimental tasks are included in Table 4.1. Example in section 2.4.5 again shows the complexity calculation for a simple experimental task, but based on HCM. Example in section 2.4.6 works a more complex experimental task through HCM.

2.4.5 Example 1: Halstead complexity metric

```
public class task2{
   public static void main(String arg[]){
```

```
        int a = 10;
        int b = 20;
        a = b;
        System.out.format("%d %d", a, b);
    }
}
```

n1	Operator	N1	n2	Operand	N2
1	public class task2 {}	1	1	arg[]	1
2	public static void main {}	1	2	a	3
3	int	2	3	b	3
4	String	1	4	”%d %d”	1
5	=	3	5	10	1
6	;	4	6	20	1
7	System.out.format	1			
8	()	2			
9	,	2			
$n_1 = 9$		$N_1 = 17$	$n_2 = 6$		$N_2 = 10$

Table 2.5: Halstead calculation example 1

Calculation:

$$n_1 = 9 \; n_2 = 6 \; n = 15$$

$$N_1 = 17 \; N_2 = 10 \; N = 27$$

$$Calculated \; program \; length \quad (H) = 44.039$$

$$Volume \quad (V) = 105.48$$

$$Difficulty \quad (D) = 7.5$$

$$Effort \quad (E) = 791.1$$

2.4.6 Example 2: Halstead complexity metric

```
public class task23{
   private static int foo(int a){
      return a*2;
   }
   public static void main(String arg[]){
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 3; i++){
         array[i] = foo(array[i]);
      }
      System.out.format("%d %d %d", array[0], array[1], array[2]);
   }
}
```

n1	Operator	N1	n2	Operand	N2
1	public class task23 {}	1	1	a	2
2	private static int foo {}	2	2	2	1
3	public static void main	1	3	arg[]	1
4	String	1	4	”%d%d%d”	1
5	for	1	5	array[]	6
6	=	4	6	output	1
7	<	1	7	i	3
8	()	4	8	0	2
9	{}	1	9	5	1
10	,	5	10	3	2
11	*	1	11	7	1
12	int	4			
13	System.out.format	1			
14	++	1			
15	;	7			
16	return	1			
$n_1 = 16$		$N_1 = 36$	$n_2 = 11$		$N_2 = 21$

Table 2.6: Halstead calculation example 2

Calculation:

$$n_1 = 16\ n_2 = 11\ n = 27$$

$$N_1 = 36\ N_2 = 21\ N = 57$$

$$Calculated\ program\ length \quad (H) = 102.053$$

$$Volume \quad (V) = 271.028$$

$$Difficulty \quad (D) = 15.27$$

$$Effort \quad (E) = 4138.59$$

2.4.7 McCabe cyclomatic complexity measures (CCM)

Cyclomatic complexity measures (CCM) were introduced by Thomas J. McCabe in 1976. In CCM, rather than counting lines or counting operator and operands, the focus is on logical structure of the program. The theorem given by [McC76] is as follows: Definition 1: The cyclomatic number V(G) of a graph G with n vertices, e edge, and p connected components is

$$V(G) = e\ -\ n\ +\ p \tag{2.11}$$

Theorem 1: In a strongly connected graph G, the cyclomatic number is equal to the maximum number of linearly independent circuits.

The sequence of an arbitrary number of nodes always has unit complexity and that cyclomatic complexity conforms to our intuitive notion of "minimum number of paths". Several properties of cyclomatic complexity are stated below: [McC76]

- V(G) ≥ 1
- V(G) is the maximum number of linearly independent paths in G

- Inserting or deleting functional statements to G does not affect V(G)
- G has only one path if and only if V(G) = 1
- Inserting a new edge in G increases V(G) by unity
- V(G) depends only on the decision structure of G

All programs should have a start point and a exit point. From the definition(2.4.7) the number of connected components is referred as p. A connected component is a node in the graph to which there is a path from the entry node and from which there is a path by which the exit node may be reached. So, when the graph is strongly connected then we use the formula

$$V(G) = e - n + p \tag{2.12}$$

and if the graph is not strongly connected then the formula

$$V(G) = e - n + 2p \tag{2.13}$$

is used.

Cyclomatic complexity measures for all experimental tasks are included in Table 4.1. Again, two examples are provided, Example in section 2.4.8 shows the CCM calculation for a simple task and Example in section 2.4.9 calculates CCM for a more complex one.

2.4.8 Example 1: McCabe cyclomatic complexity

```
public class task2{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      a = b;
      System.out.format("%d %d", a, b);
   }
}
```

Calculation:

$$E = 2,\ N = 3,\ P = 1$$

$$According\ to\ formula:\ V(G) = E - N + 2P$$

$$V(G) = 2 - 3 + 2 * 1$$

$$V(G) = 1$$

2.4.9 Example 2: McCabe cyclomatic complexity

```
\begin{BVerbatim}
public class task23{
   private static int foo(int a){
      return a*2;
   }
   public static void main(String arg[]){
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 3; i++){
         array[i] = foo(array[i]);
      }
      System.out.format("%d %d %d", array[0], array[1], array[2]);
   }
}
```

Calculation:

$$E = 7,\ N = 9,\ P = 3$$

$$According\ to\ formula:\ V(G) = E - N + 2P$$

$$V(G) = 7 - 9 + 2 * 3$$

$$V(G) = 4$$

CHAPTER 3

Methodology

This chapter describes the necessary steps for generating the results of this study.

In general, participants perform a set of 25 tasks with varying difficulty. They are given one code snippet per task, for which they must mentally compute the output. For all participants, tasks are presented in Java, since it is the programming language taught in all introductory computer science courses at SIUE.

Firstly, the design of of the experimental tasks is described (3.1), followed by instrumentation (3.2) and experimental procedure (3.3). Lastly, signal analysis is discussed (3.4). The complete process can be from the diagram as shown in figure 3.1.

3.1 Experimental Tasks

Some of the 25 experimental tasks used for this study are based on the tasks presented in the work by Bornat and Dehnadi [BDS08]. The tasks constructed by Dehnadi were intended to find strong predictors of success in learning programming; we extended these tasks by constructing ones with additional variable assignment operations, arrays, loops, conditionals, and function calls. Altogether, there were 25 tasks which were presented to each participant with the help of an automated task delivery system. All tasks were written in Java, because it is the primary instruction language at the participants' institution, i.e. Southern Illinois University Edwardsville. The tasks may be summarized as follows:

- **Task 1:** This task consisted of reading a paragraph of English prose;
- **Task 2 - Task 4:** Single-variable assignment operations involving two variables and expected to be the simplest of the tasks in the experiment, with variations in ordering of variable assignments and and naming of variables;

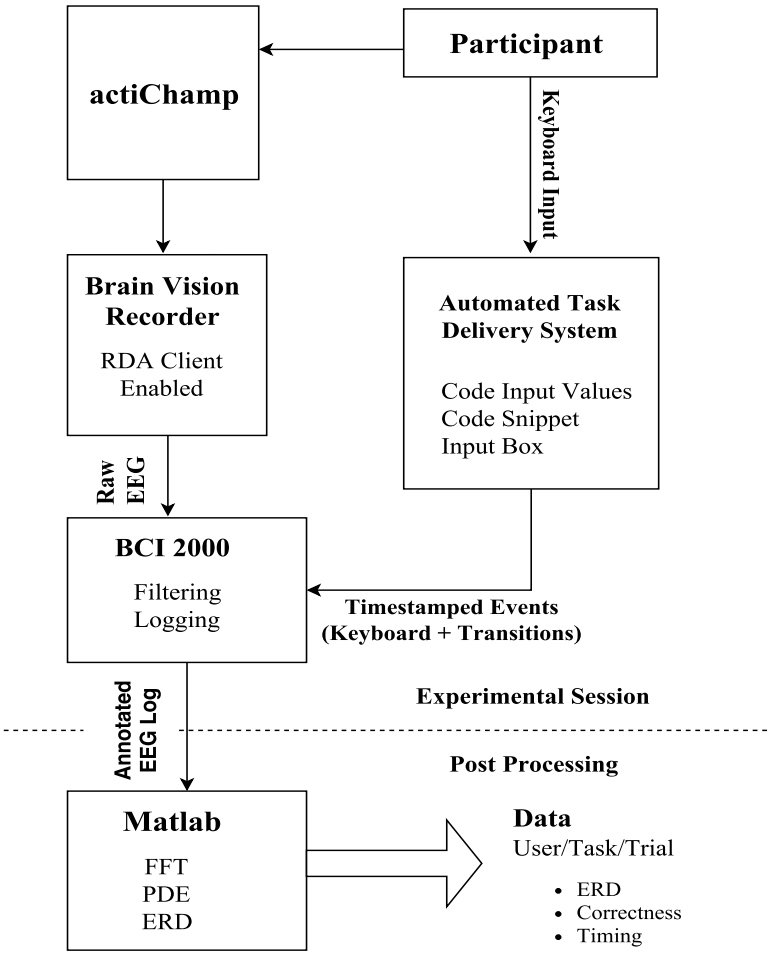

Figure 3.1: *Process.* Complete experimental process from collecting data to analyzing it

- **Task 5 - Task 7:** Two-variable assignment operations involving two or three variables with variations in ordering of variable assignments;
- **Task 8 - Task 13:** Three-variable assignment operations with three variables and variations in the ordering of variable assignments;
- **Task 14 - Task 16:** Three-variable assignment operations with three variables, variations in the ordering of assignments and assignments depending on conditionals;
- **Task 17 - Task 22:** Array variable assignments and conditional assignments with loops, nested loops and some computation +, *;

- **Task 23 - Task 25:** Array variable assignments and conditional assignments with loops, nested loops, some computation +, *, simple and complex function calls.

3.2 Instrumentation

Raw EEG data was captured by BrainVision's actiChamp amplifier, a research-grade amplifier with 32 electrodes and capable of sampling rates up to 10kHz. The software used included BrainVision's Recorder, BCI2000, and the Automated Task Delivery System. The Recorder delivers the raw EEG signal to BCI2000, which annotates it with keystrokes and timestamps. Integration of actiChamp and BCI2000 limited the effective sampling rate to 500Hz. The Automated Task Delivery System was developed in the lab specificially for this and related studies at SIUE and likewise is made to communicate experimental data to BCI2000. Participants were fitted with a cap containing the 32 electrodes (IRB approval was granted for the subject with approval number 16-0920-1), which keeps the electrodes relatively well fixed to the scalp, reducing noise from physical movement. The electrode placement is seen in Figure 3.2. Conductive gel is injected through the electrodes.

3.3 Procedure

Each experimental session was sub-divided into three main events: the pre-experimetnal session (3.3.1), the experiment (3.3.2), and the post-experimental session (3.3.3). Each is described below:

3.3.1 Pre-experimental session

In the pre-experimental session the participant receives a brief introduction to the EEG device, the software used during the experiment, and a description of the procedures encountered during the experiment. The participants were shown the cap fitted with electrodes and shown how the electrolyte gel is to be added through the electrodes to

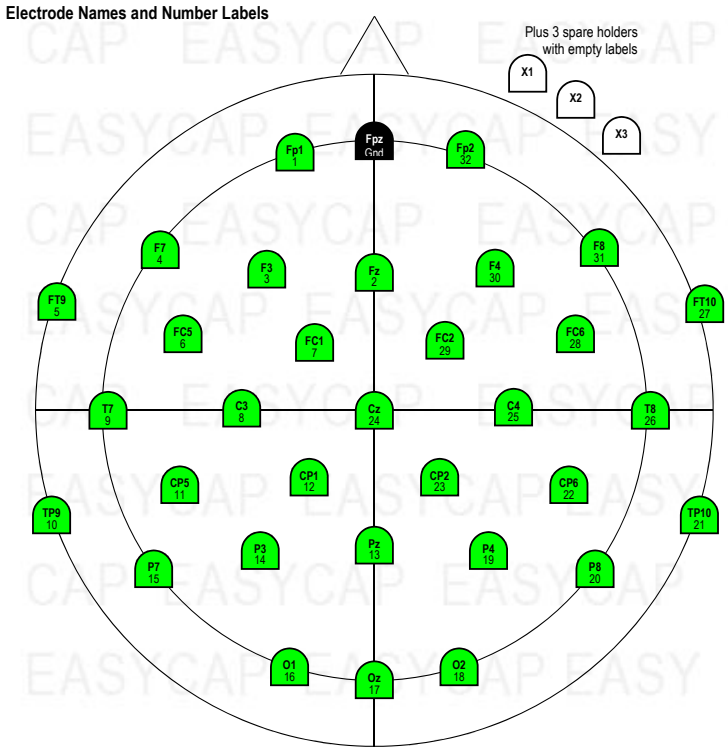

Figure 3.2: *Electrode position.* 32 data channel provided by the actiChamp are labeled by their placement

ensure a reliable connection between the scalp and electrodes which are shown in the figure 3.3

A pilot study indicated that without an overview of the experiment, or tutorial, participants become confused and have trouble entering the expected response correctly. As a result, we briefly instruct the participant about the flow of experiment, the programming language used, and the format of expected responses. The operation of the Automated Task Delivery System was also described, with special attention given to the rest screen (e.g. screen displaying the fixation cross) shown in Figure 3.4, to emphasize that they should rest and relax during this time. Additionally, the format of the task screen, see Figure 3.5, and the automated response to user input (screen showing whether the response was correct or incorrect).

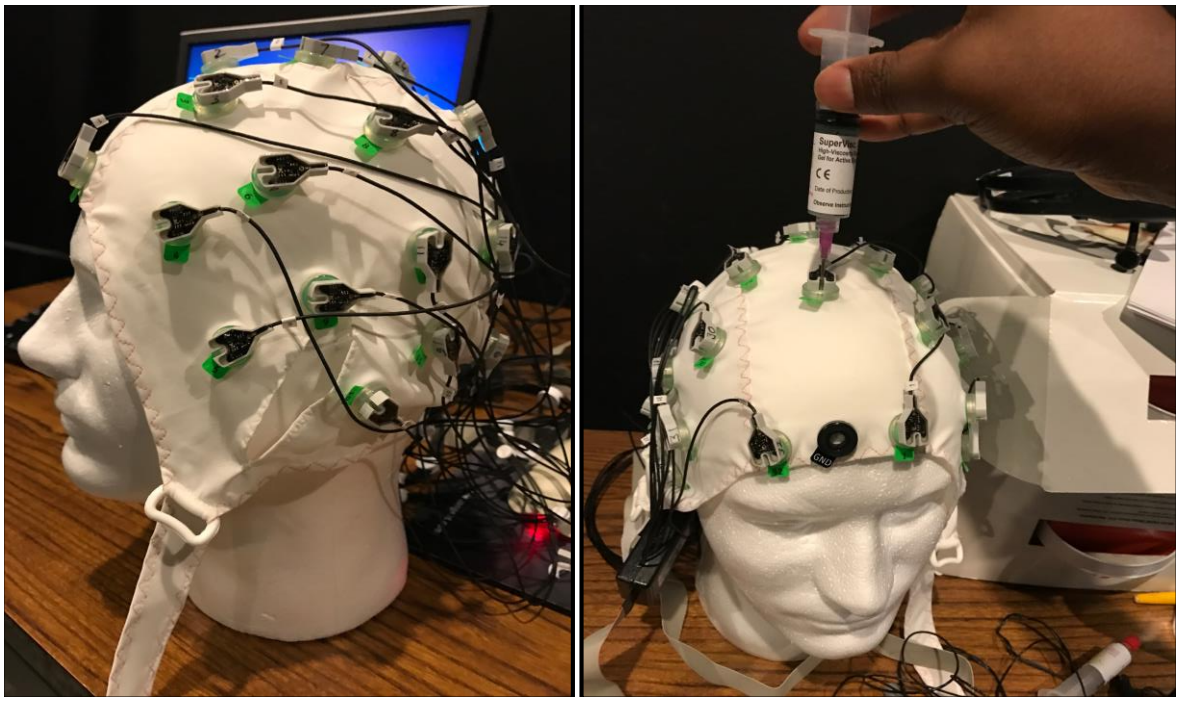

Figure 3.3: *Cap fitted with electrodes.* Cap fitted with electrodes and how the electrolyte gel is added through the electrodes to ensure reliable connection between scalp and the electrode

Participants were instructed that the rest screen was followed by the task screen containing task, text box for input answer and buttons either to submit or skip which is shown in Figure 3.5.

EEG is very sensitive to noise from muscle movement which has an adverse affect on the recorded signal. So each participant wes instructed to be calm, i.e. refrain from excessive movement of their arms, legs, head, jaws (basically any kind of muscle movement). To make the participant comfortable and reduce muscle movement the task presentation program is fully automated and keyboard driven so that the participant can rest their arm on table in a position that allowed them to reach the keyboard without excessive movement.

Following the aforementioned instruction, each participant was provided two forms: 1)

Figure 3.4: *AutoTask program rest screen.* This figure shows rest screen shown by the automated task program which delivers tasks for the participant

informed consent form which explains the purpose of the project, risk factors, benefits to science and the confidentiality of records. And 2) Pre-Survey form where the general questions were asked such as: age, sex, major, years in programming, professional experience in programming, approximate size of the largest project worked on, an estimate of overall programming experience, Java experience, programming experience compared to class mates, and programming experience compared to experts with 20 years of practical experience. The response from the participant is shown in the table 3.1.

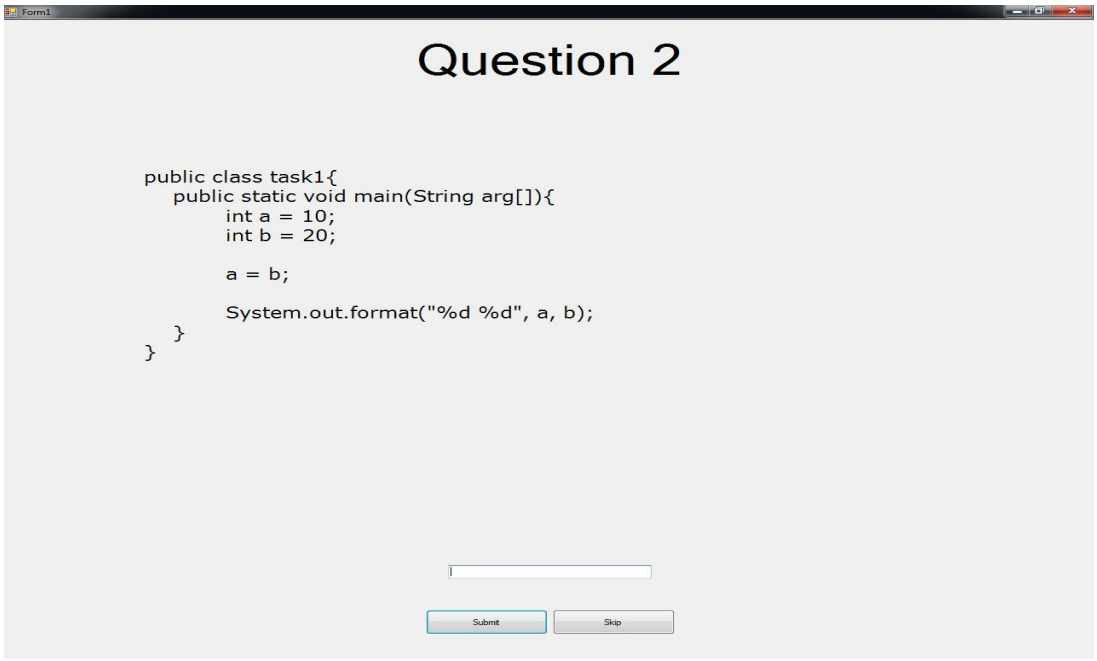

Figure 3.5: *AutoTask program task screen.* This figure shows task screen presented to participant by the AutoTask program for delivering the the programming tasks for the participant

No.	Age	Sex	Major	First Enrolled	Coding Courses	Largest Project LOC	Num. Lang. Known	Experience					
								Professional (yrs.)	Programming (yrs.)	Overall	Java	vs Peer	vs Expert
1	20	F	CS	2015	2	100	2	No	1.5	3	4	3	2
2	22	M	CS	2014	4	200	3	No	2	3	2	3	1
3	22	M	CS	2012	9	400	7	Yes	6.5	3	3	4	2
4	19	M	CS	2015	4	600	3	No	3	4	4	4	1
5	21	M	CS	2014	6	1000	4	No	3	5	3	4	2
6	21	F	CS	2015	N/A	200	1	N/A	2	3	3	4	1
7	20	M	CS	2015	5	500	3	No	3	3	4	3	1
8	21	M	CS	2015	6	1000	4	No	8	3	4	3	1
9	21	F	CE	2014	4	250	1	No	3	3	4	3	2

Table 3.1: *Participant pre survey.* No. (#) represents the participant number. For Overall programming experience and java experience it is encoded with {1: Very Inexperienced, 2:Inexperienced, 3: Medium, 4:Experienced, 5:Very Experienced}, Similarly for programming experience compared to class and programming experience compared to expert with 20 years of practical experience is encoded as {1: Much Worse, 2:Worse, 3: The Same, 4:Better, 5:Considerably better}

3.3.2 Experimental session

Experimental session was designed to last no more than 30 minutes so that the participant do not get overwhelmed. The experiment was conducted in a normal environment consisting of office, chair, desk, and computer within a shielded room containing minimal distractors i.e. exhibiting impreceptible levels of ambient noise and containing black painted walls for no adornments.

In this module the cap was fitted on the participant head and high viscosity electrolyte gel for active electrodes were injected through the electrode with the help of a syringe. The gel ensures a reliable connection between the scalp and the active electrode. Active electrodes were used because it is safe and high dry skin impedance can be omitted by using amplifier with very high input impedance. Brain Vision Recorder software was started to check the impedance of each channel to assure good signal quality, i.e. signal to noise ratio, and to activate the RDA client so that BCI2000 can get access to the raw EEG signal. The Automated Task Delivery System was used to present the experimental tasks and tag and timestamp relevant events and keystrokes in the EEG dataset.

After starting, the Automated Task Delivery System delivers the rest screen first so that participant can relax and prepare for the upcoming task. The time for the rest screen is set to 10 seconds and it provides the reference point for measuring the EEG signal power during the task. The experiment was not timed to relieve any pressure that may affect a participants performance. Also the answer verification screen was presented with feedback whether the answer provided by the participant was "Correct", "Incorrect" or "Skipped".

3.3.3 Post-experimental session

Following the experiment, each participant was given a survey based on the NASA-TLX survey, modified to include only the metrics relevant to this study. NASA-TLX

is a multi-dimensional scale designed to obtain workload estimates from one or more operators while they are performing a task immediately afterwards [Har06]. In this study the participants were only asked to report their mental demand and stress level while solving the tasks during the experiment and it was based on NASA-TLX. An example item from the modified NASA-TLX survey is shown in

```
public class task1{
    public static void main(String arg[]){
        int a = 10;
        int b = 20;

        a = b;

        System.out.format("%d %d", a, b);
    }
}
```

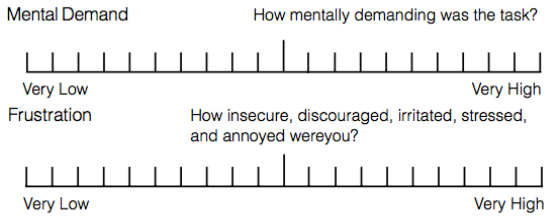

Figure 3.6: *Modified NASA-TLX*. Sample item from the modified NASA-TLX survey.

3.4 Signal Analysis

The participant's EEG is continuously recorded for the duration of the experimental session. The raw EEG signal is broken up into individual rest and trial sections, representing each of the 25 experimental task trials and their preceding rest periods. Each of the rests and trials are tagged and timestamped by the Automated Task Delivery System during the experiment.

The signal was recorded using all 32 channels but specific 8 channels ware selected for signal processing. Based on fMRI study [RTJ+00] we used electrodes: Fp1, Fz, F3, F7, FT9, FC5, FC1 and C3 which are clustered near Brodmann areas 8 and 46 found in the prefrontal cortex, which are confirmed to correspond to working memory activity. Figure (3.2) shows the position of electrodes on.

To perform the analysis, the raw EEG signal was processed with Matlab (Version R2014b), using, in part, features from the EEGLAB and Fieldtrip EEG analysis toolboxes. The raw EEG signal was first separated to states (holds information such as time, task,

rest, trial, input etc.), parameters (holds information about sampling rate, subject name, source channels, channels name etc.), signal and total sample (size of the samples). For this study the interested signal frequency ranges within 2Hz to 15Hz i.e. frequency range containing Theta and Alpha waves. The signal for range of 2Hz - 15Hz was adjusted by the help of band-pass filter. Signal was normalized after the adjustment of the signal to the mentioned frequency range and it was done by finding the mean of the signal and subtracting from the total signal. Welch [Wel67] method was used to estimate the Power Spectral Density (PSD).

Alpha frequency is further subdivided into the three sub-bands as described in section 2.2 (Lower 1 Alpha (L1A), Lower 2 Alpha (L2A), and Upper Alpha (UA)), the ranges of which are based on the Individual Alpha Frequency (IAF). The theta wave was also included for analysis because it is expected to have an inverse relationship with alpha wave (when alpha synchronizes, theta is expected to desynchronize). Figure 3.7 shows the Welch's PSD estimate of the rest versus trial state for the all participant with accurate response to the tasks. The IAF is indicated by a circle, and vertical lines sepereate the four sub-bands.

ERD was calculated using Equation 2.3. Each rest and trial period was split into 125ms window frames. The power of rest period for each alpha sub-band is computed for reference and averaged over the windows to yield the baseline rest power shown in equation 2.3. Sub-band power is similarly computed for the trial period, and finally, ERD is computed.

Each data point was also tagged with an error rate. Any reading outside of the bounds of $-200\mu V$ to $+200\mu V$ exhibit too much noise to be considered a good measure of brain activity. The error rate for each epoch is the percentage of samples that are outside of that range. Any rest or trial period with error rate higher than 20% is considered noisy [Kli99] and can be excluded from the analysis.

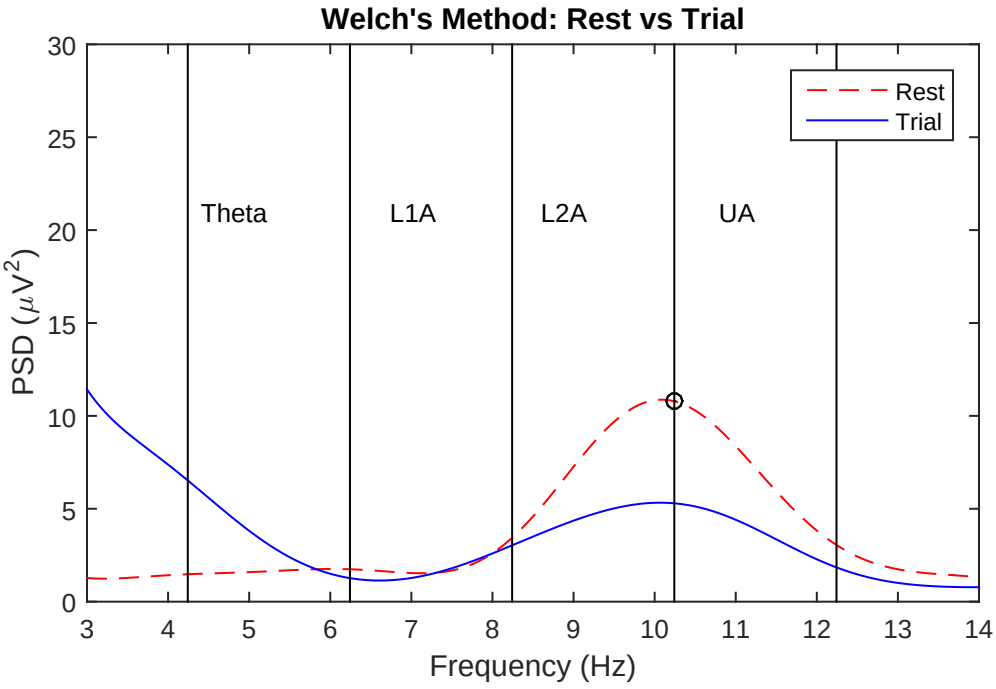

Figure 3.7: *Pwelch rest vs trial.* Shows Power Spectral Density between the rest and trial states. The IAF is marked by a circle, and the resulting sub-band ranges are indicated as Theta, Lower 1 Alpha, Lower 2 Alpha, and Upper Alpha

CHAPTER 4

Results

This chapter describes the findings after the experiment. We calculated the complexity, length, and effort of the task with classical software complexity metrics (Halstead, McCabe, PSLOC, and LSLOC), we also processed the recorded EEG signal. Linear regression analyses with ANOVA were performed on the processed data to assess whether the participants' effort on the experimental tasks is seen in the Upper Alpha, Lower 2 Alpha, Lower 1 Alpha, and Theta EEG bands. Additionally, we record task duration directly from the task presentation system, and following the experiment conduct a modified NASA-TLX survey to obtain mental demand and stress levels. The analyses are intended to test whether we are observing noise or a potential indicator of effort based on the difficulty of a task.

The rest of the chapter is followed in the following order: First, the computation of tasks with classic software complexity metric (4.1), followed by correct response for each task from the participant (4.2), linear regression analysis with ANOVA along with bar charts for all the experimental tasks (4.3), and finally, we manually reduce the number of task categories from 25 to 2, by classifying the existing tasks as either easy or hard, based on their Halstead complexity for effort, then perform an additional analysis on these two groups (4.4).

4.1 Computation with Complexity Metric

Complexity measurements were performed on all tasks, except for the English prose were computed by hand, using the various classic software complexity metrics: Lines of Code (LOC), Halstead's metrics, and McCabe's cyclomatic complexity. The results are shown in Table 4.1. Complexity measurements provide us with a base understanding of the difficulty levels of our experimental tasks, which aids in the interpretation of results

observed from the EEG. As shown in the table, the Halstead complexity metric appears to show more variety in the calculated values, indicating that perhaps it has a finer granularity than the other metrics. In any case, the results obtained from the classical software complexity metric computation shows the tasks have varying difficulty.

Task	**Lines of Code**		**Halstead**					**McCabe**
	PSLOC	**LSLOC**	**H**	**V**	**D**	**E**	**T**	**V(G)**
2	10	6	44.039	105.48	7.5	791.1	43.95	1
3	10	6	44.039	105.48	7.5	791.1	43.95	1
4	10	6	44.039	105.48	7.5	791.1	43.95	1
5	11	7	44.039	121.11	9	1089.99	60.55	1
6	11	7	44.039	121.11	9	1089.99	60.55	1
7	12	8	52.529	155.32	8.43	1310.43	72.80	1
8	13	9	52.529	171.67	9.56	1641.16	91.17	1
9	13	9	52.529	171.67	9.56	1641.16	91.17	1
10	13	9	52.529	171.67	9.56	1641.16	91.17	1
11	13	9	52.529	171.67	9.56	1641.16	91.17	1
12	13	9	52.529	171.67	9.56	1641.16	91.17	1
13	13	9	52.529	171.67	9.56	1641.16	91.17	1
14	16	10	62.053	199.65	13.06	2607.429	144.85	2
15	16	10	62.053	199.65	13.06	2607.429	144.85	2
16	16	10	62.053	203.90	13.06	2663.44	147.96	2
17	12	6	67.75	177.19	13	2303.47	127.97	2
18	13	7	96.32	230.32	11.66	2685.53	149.19	2
19	14	8	112.50	281.76	16.29	4589.87	254.99	3
20	16	8	86.52	265.92	15.4	4095.16	227.51	3
21	17	9	86.52	284.26	17.5	4974.55	276.36	3
22	19	9	92.52	264.70	16	4235.2	235.28	4
23	17	9	102.053	271.028	15.27	4138.59	229.92	4
24	17	9	113.11	340.05	18.81	6396.34	355.35	4
25	20	12	128.09	415	18.13	7523.95	417.99	4

Table 4.1: *Calculated code complexity.* Code complexity calculations using classic software complexity measures. PSLOC = Physical Source Lines of Code, LSLOC = Logical Source Lines of Code, H = Calculated Estimated Program Length, V = Volume, D = Difficulty, E = Effort, T = Estimated time to solve the program (using Stroud number 18)

4.2 Correct Response for the Tasks

We had nine participants for the experiment where each of them was provided with 25 tasks. Figure 4.1 shows the number of participants that performed the task correctly. Tasks 24 and 25 were correctly answered by only one participant; these tasks are hence omitted from further analysis.

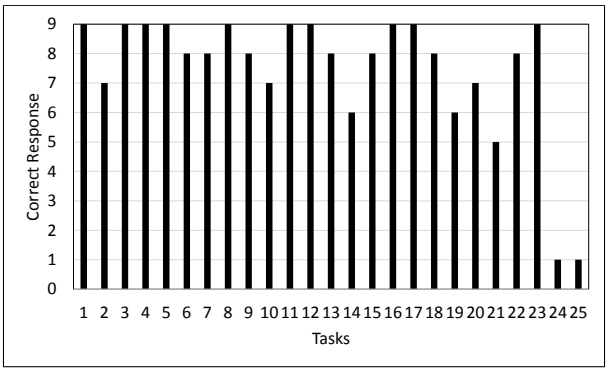

Figure 4.1: *Correct response for tasks.* Shows the correct response from the nine participants for solving the tasks

4.3 Regression Analysis: All Tasks Individually

After processing the raw EEG signal with MATLAB, we perform statistical analyses using IBM SPSS (version 24). Specifically, we performed linear regressions of task difficulty, as represented by the set of experimental tasks sorted by Halstead complexity for effort, and the ERD measured within each sub-band: UA 4.2, L2A 4.3, L1A 4.4, Theta 4.5. Additionally, we perform an analysis based on task difficulty and duration 4.8, mental demand 4.6, and stress level 4.7 As already stated, tasks were sorted by increasing difficulty based on the tasks' Halstead's complexity for effort and task number was used as an independent variable. Additionally, we ignored tasks which were skipped or completed incorrectly; it is only for the correct responses that we can be certain that the

participant paid attention and performed the cognitive tasks necessary to complete the coding comprehension task. Table 4.2 shows the output of linear regression based on the variables described above, filtering out incorrect and skipped tasks and ignoring tasks 1, 24, and 25 (english prose, and the two tasks that were correctly answered by a single participant).

Variables	*df*		**F**	**Sig.**	R^2	**Intercept**	**Slope**
	regression	residual					
UA	1	173	4.050	.046	.023	33.244	.613
L2A	1	173	8.051	.005	.044	21.243	.767
L1A	1	173	3.419	.066	.19	9.336	.308
Theta	1	173	.948	.332	.005	12.747	.149
Duration	1	173	75.323	.000	.303	.588	.849
SMD	1	173	104.395	.000	.376	2.517	1.611
SSL	1	173	19.355	.000	.101	8.654	.918

Table 4.2: *Linear regression analysis.* Linear regression analysis for the variables UA = Upper Alpha, L2A = Lower 2 Alpha, L1A = Lower 1 Alpha, Theta, Duration, SMD = Survey Mental Demand, and SSL = Survey Stress Level

Simple linear regressions were calculated to predict UA, L2A, L1A, and Theta desynchronization, duration, mental demand and stress levels across the tasks. A significant regression equation was found ($F(1, 173) = 4.050, p < .046$), with an R^2 of .023 and participants' predicted UA is equal to $33.244 + .613$ (task number), when UA is measured in microvolts (μV). UA increased by $.613\mu V$ for each successive task.

Similarly, A significant regression equation was found ($F(1, 173) = 8.051, p < .005$), with an R^2 of .044 and participants' predicted L2A is equal to $21.243 + .767$ (task number), when L2A is measured in microvolts (μV). L2A increased by $.767\mu V$ for each successive task.

A significant regression equation was found ($F(1, 173) = 75.323, p < .000$), with an R^2 of .303 and participants' predicted duration is equal to $.588 + .849$ (task number), when duration is measured in milliseconds. Duration increased by .849 milliseconds for each

successive task.

A significant regression equation was found ($F(1, 173) = 104.395, p < .000$), with an R^2 of .376 and participants' predicted survey mental demand is equal to $2.517 + 1.611$ (task number), when survey mental demand is measured in scale of 20. Survey mental demand increased by 1.611 scale for each successive task.

A significant regression equation was found ($F(1, 173) = 19.355, p < .000$), with an R^2 of .101 and participants' predicted survey stress level is equal to $8.654 + .918$ (task number), when survey stress level is measured in scale of 20. Survey stress level increased by .918 scale for each successive task.

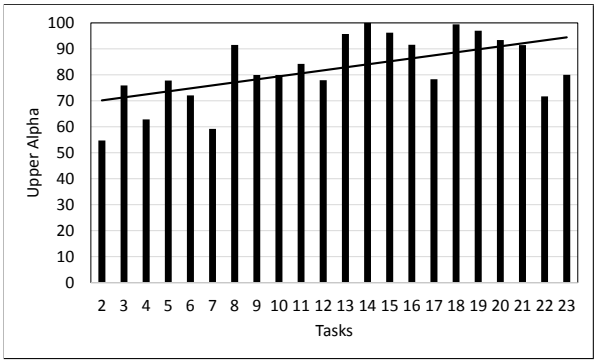

Figure 4.2: Task vs upper alpha

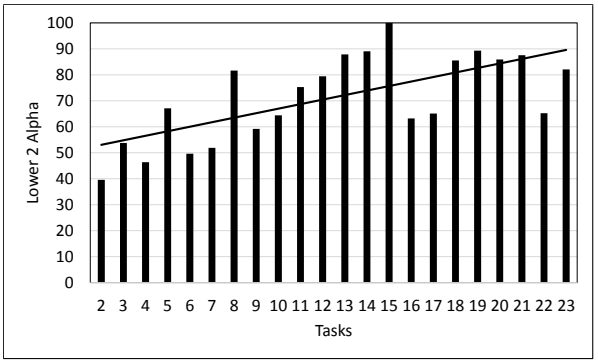

Figure 4.3: Task vs lower 2 alpha

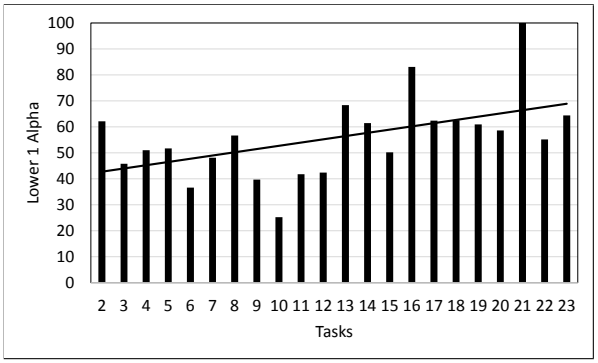

Figure 4.4: Task vs lower 1 alpha

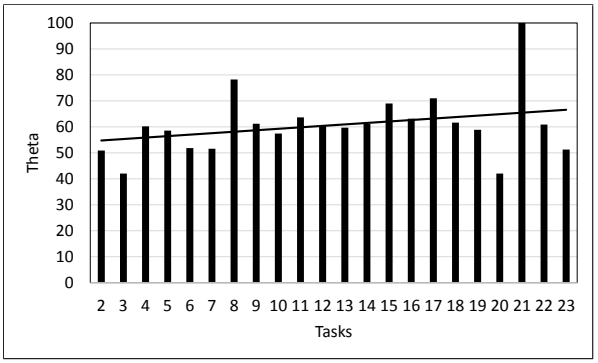

Figure 4.5: Task vs theta

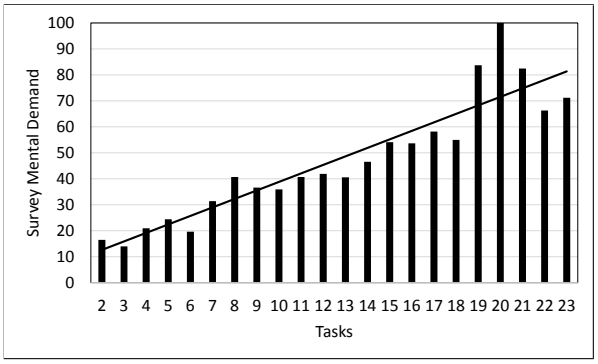

Figure 4.6: Task vs survey mental demand

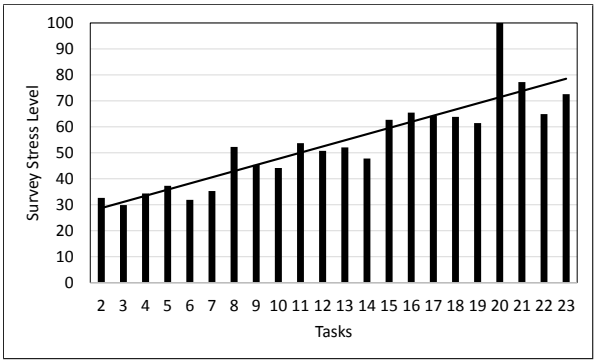

Figure 4.7: Task vs survey stress level

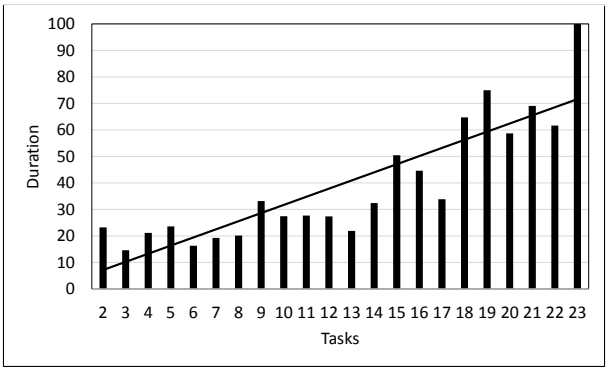

Figure 4.8: Task vs duration

4.4 Regression Analysis: Easy vs Hard Tasks

The experimental tasks were recoded into easy and hard tasks Simple linear regressions were calculated to predict UA, L2A, L1A, and Theta desynchronization, duration, mental demand and stress levels across the encoded easy and hard tasks. A significant regression equation was found ($F(1, 173) = 4.640, p < .033$), with an R^2 of .026 and participants' predicted UA is equal to $28.705 + 8.120$ (task number), when UA is measured in microvolts (μV). UA increased by $8.120 \mu V$ for each successive task.

Similarly, A significant regression equation was found ($F(1, 173) = 6.764, p < .010$), with an R^2 of .038 and participants' predicted L2A is equal to $17.648 + 8.744$ (task number), when L2A is measured in microvolts (μV). L2A increased by $8.744 \mu V$ for each successive task.

Variables	*df*		**F**	**Sig.**	R^2	**Intercept**	**Slope**
	regression	residual					
UA	1	173	4.640	.033	.026	28.705	8.120
L2A	1	173	6.764	.010	.038	17.648	8.744
L1A	1	173	5.738	.018	.032	5.826	4.912
Theta	1	173	.516	.474	.003	12.542	1.362
Duration	1	173	47.009	.000	.214	-2.139	8.822
SMD	1	173	62.627	.000	.266	-2.691	16.767
SSL	1	173	13.379	.000	.072	5.616	9.602

Table 4.3: *Linear regression analysis.* Linear regression analysis for the variables UA = Upper Alpha, L2A = Lower 2 Alpha, L1A = Lower 1 Alpha, Theta, Duration, SMD = Survey Mental Demand, and SSL = Survey Stress Level

Similarly, A significant regression equation was found ($F(1, 173) = 5.738, p < .018$), with an R^2 of .032 and participants' predicted L1A is equal to $5.826 + 4.912$ (task number), when L1A is measured in microvolts (μV). L1A increased by $4.912\mu V$ for each successive task.

A significant regression equation was found ($F(1, 173) = 47.009, p < .000$), with an R^2 of .214 and participants' predicted duration is equal to $-2.139 + 8.822$ (task number), when duration is measured in milliseconds. Duration increased by 8.822 milliseconds for each successive task.

A significant regression equation was found ($F(1, 173) = 62.627, p < .000$), with an R^2 of .266 and participants' predicted survey mental demand is equal to $-2.691 + 16.767$ (task number), when survey mental demand is measured in scale of 20. Survey mental demand increased by 16.767 scale for each successive task.

A significant regression equation was found ($F(1, 173) = 13.379, p < .000$), with an R^2 of .072 and participants' predicted survey stress level is equal to $5.616 + 9.602$ (task number), when survey stress level is measured in scale of 20. Survey stress level increased by 9.602 scale for each successive task.

CHAPTER 5

Discussion

5.1 Interpretation of Results

Analysis of the experimental data, as shown in the previous chapter, assumes that for the experimental tasks, we would observe a range of cognitive effort. Prior results have shown, for example, that power in the alpha band increases (synchronizes) when a subject is relaxing, and decreases (desynchronizes) in the presence of mental demand. It was also previously shown that alpha desynchronization is proporitional to the complexity or difficulty of a task, to some extent. It is therefore reasonable to assume that given a set of tasks ranging in complexity, we would likewise observe a range of cognitive effort. What was not known is how sensitive EEG-based cognitive load measurement is to small changes in task complexity. To a large extent, it is still unknown, pending an experiment with a much larger set of participants. However, certain features are evident even with a small set of subjects. For example, based on our statistical analysis we find that the relationship between desynchronization in the Upper Alpha, Lower-2 Alpha, and Lower-1 Alpha sub-bands and tasks ordered by their Halstead complexity is significant, but with low R^2, as listed in Table 4.2. This result indicates that we are likely observing consistently increasing levels of desynchronization within the UA, L2A, L1A sub-bands. With respect to cognitive load theory, this means that as the task difficulty increases the demand for working memory while performing task correctly also increases. This is corroborated by the Halstead complexity calculations, since those take into account the number of operands in the task.

Given that our regression models are significant, but have low R^2 values, the models themselves are likely poor predictors of the exact cognitive demands for a given task. Looking to find a model that improves the prediction of a task's cognitive demand given

some EEG-based observation of effort, we simplify the task set by grouping tasks into a set of easy tasks and a set of hard tasks, again using the Halstead complexity metric to split the tasks. What we find is that the model comparing EEG-based measures of cognitive load on easy and hard tasks to the computed Halstead complexity for effort of the tasks themselves results in another significant model with a slight improvement to the model's predictive power.

What we can assume is that 9 participants do not provide us with enough data points to compute a strongly predictive model. The relationship exists, but at this time there is insufficient data to make strong predictions of cognitive effort from the tasks themselves or accurately rate a task's difficulty based on EEG readings alone.

5.2 Threats to Validity

Human brain function varies from person to person. When accounting for measuring effort in humans, it is important to recognize that this kind of study has its own limitations and ours is no exception. In this section, we briefly discuss some immediate threats to the validity of our experiment and analyses.

First, EEG is highly sensitive to signal artifacts from muscle movement, which is briefly explained in the above sections. In this study, participants are not restrained; instead, the participants are encouraged to limit movement, and indeed, the experiment is designed to minimize movement through minimal interaction. Participants have to use a keyboard for provide answers for each task. Additionally, head and facial movements could have occurred which were captured along with the relevant parts of the EEG signal. While we attempt filtering out this muscle movement noise, it can never be fully eliminated from the signal.

Second, we raise the possibility that the modified NASA-TLX survey results may be subject to bias. The survey is performed after the experiment is completed, and all tasks are presented together, three or four tasks per page on successive printed pages.

The NASA-TLX survey items were presented alongside the task that they relate to. It is possible that the ordering of the tasks themselves, both in the experiment and the subsequent, similar ordering of the survey, indicated to the participant that the tasks ought to be increasing in complexity and/or difficulty and may have guided the participants to rate them as such, influencing the strong statistical relationship of the survey to the task complexity.

Third, we tried to control for expertise by limiting our recruitment of subjects and actual participants to the same class level (the participants were currently taking CS 240). Moreover, our assumption was that self-reported experience data is unreliable (for example, some people report years of experience based on exposure to a concept). It is possible that there is a wider variety of experience even at this early stage of study in the CS program at SIUE which this study does not account for.

Fourth, for this experiment, we rely on 25 programming tasks of increasing complexity (based on Halstead's complexity). There are clearly groups of tasks for which there are no differences in Halstead complexity. It is unknown whether these tasks should actually produce different levels of cognitive effort or not. It is clear from our results that comparisons of our obsevations with calculations of Halstead's effort, a relationship exists, but it clearly is not telling the whole story.

CHAPTER 6

Future Works

This work demonstrates the potential applicability of EEG-based measurement for cognitive load. Primarily, this would could benefit from increasing the number of study participants to strengthen the statistical analyses. Future work could immediately include a follow-up study that would generate additional data. Subsequently, there are other several other research possibilities based on the this work.

First, classical software complexity metric provides approximate predictions of effort which may be supplemented by EEG-based measures of cognitive load. Predictions of effort made based on classical complexity measures as well as personalized EEG-based measures, could result in new methods of software project sizing.

Second, studies like this one may make it possible to fine-tune the sequencing of concepts in a programming curriculum. For example, understanding the cognitive load of assignment operations, conditionals, loops, function calls, etc., and understanding how this cognitive load changes with increasing expertise, creates the possibility of creating a curriculum that optimizes for learnability, based on the the practice of cognitive load theory. In effect, examples and programming assignments might be sequenced such that they do not overload the working memory of the student, rather build on previously learned concepts.

Finally, we think that this study is a step towards the future of understanding how the programmer perceives a programming task. Further research may be done to construct more effective programming language designs.

CHAPTER 7

Conclusion

The main purpose of this study is to show that a range of programmer effort can be measured by the use of the EEG. Based on our methodology and findings, we observed that desynchronization within the UA, L2A, and L1A sub-bands of the alpha frequency range is related to task complexity, indicating that working memory demand increases with task difficulty. This study raises the possibility of future studies approximating programmer effort, fine-tuning curriculum sequencing, language design, and program sizing.

REFERENCES

[BAC00] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approachesa survey. *Annals of software engineering*, 10(1-4):177–205, 2000.

[Bad92] Alan Baddeley. Working memory and conscious awareness. In *Theories of memory*, pages 11–20. Lawrence Erlbaum Associates, 1992.

[Bak07] Stuart N Baker. Oscillatory interactions between sensorimotor cortex and the periphery. *Current opinion in neurobiology*, 17(6):649–655, 2007.

[BDS08] Richard Bornat, Saeed Dehnadi, and Simon. Mental models, consistency and programming aptitude. In *Proceedings of the tenth conference on Australasian computing education-Volume 78*, pages 53–61. Australian Computer Society, Inc., 2008.

[Ber29] Hans Berger. Über das Elektroenkephalogramm des Menschen. *Archiv für Psychiatrie und Nervenkrankheiten*, 87:527–570, 1929.

[CBCG06] Nicholas R Cooper, Adrian P Burgess, Rodney J Croft, and John H Gruzelier. Investigating evoked and induced electroencephalogram activity in task-related alpha power increases during an internally directed attention task. *Neuroreport*, 17(2):205–208, 2006.

[Cow01] Nelson Cowan. Metatheory of storage capacity limits. *Behavioral and brain sciences*, 24(1):154–176, 2001.

[Hal77] Maurice Howard Halstead. *Elements of software science*, volume 7. Elsevier New York, 1977.

[Har06] Sandra G Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage Publications Sage CA: Los Angeles, CA, 2006.

[HPS02] J Allan Hobson and Edward F Pace-Schott. The cognitive neuroscience of sleep: neuronal systems, consciousness and learning. *Nature Reviews Neuroscience*, 3(9):679–693, 2002.

[JGKL02] Ole Jensen, Jack Gelfand, John Kounios, and John E Lisman. Oscillations in the alpha band (9–12 hz) increase with memory load during retention in a short-term memory task. *Cerebral cortex*, 12(8):877–882, 2002.

[Kli96] Wolfgang Klimesch. Memory processes, brain oscillations and eeg synchronization. *International journal of psychophysiology*, 24(1):61–100, 1996.

[Kli99] Wolfgang Klimesch. EEG alpha and theta oscillations reflect cognitive and memory performance: a review and analysis. *Brain research. Brain research reviews*, 29(2-3):169–95, April 1999.

[KSH07] Wolfgang Klimesch, Paul Sauseng, and Simon Hanslmayr. Eeg alpha oscillations: the inhibition–timing hypothesis. *Brain research reviews*, 53(1):63–88, 2007.

[KSSW90] Lloyd Kaufman, Barry Schwartz, Carlo Salustri, and Samuel J. Williamson. Modulation of Spontaneous Brain Activity during Mental Imagery. *Journal of cognitive neuroscience*, 2(2):124–32, January 1990.

[McC76] Thomas J McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[Mil56] George A. Miller. The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63(2):81–97, 1956.

[NDRTB07] Vu Nguyen, Sophia Deeds-Rubin, Thomas Tan, and Barry Boehm. A sloc counting standard. In *COCOMO II Forum*, volume 2007, pages 1–16, 2007.

[Paa92] Fred G Paas. Training strategies for attaining transfer of problem-solving skill in statistics: A cognitive-load approach. *Journal of educational psychology*, 84(4):429, 1992.

[PP59] Lloyd Peterson and Margaret Jean Peterson. Short-term retention of individual verbal items. *Journal of experimental psychology*, 58(3):193, 1959.

[RMT07] Tonia A Rihs, Christoph M Michel, and Gregor Thut. Mechanisms of selective inhibition in visual spatial attention are indexed by α-band eeg synchronization. *European Journal of Neuroscience*, 25(2):603–610, 2007.

[RTJ+00] James B Rowe, Ivan Toni, Oliver Josephs, Richard SJ Frackowiak, and Richard E Passingham. The prefrontal cortex: response selection or maintenance within working memory? *Science*, 288(5471):1656–1660, 2000.

[SAK11] John Sweller, Paul Ayres, and Slava Kalyuga. *Cognitive Load Theory*. Explorations in the Learning Sciences, Instructional Systems and Performance Technologies. Springer New York, 2011.

[SK08] Paul Sauseng and Wolfgang Klimesch. What does phase information of oscillatory brain activity tell us about cognitive processes? *Neuroscience and biobehavioral reviews*, 32(5):1001–13, July 2008.

[Wel67] Peter D. Welch. The use of fast fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *Audio and Electroacoustics, IEEE Transactions on*, 15(2):70–73, Jun 1967.

APPENDIX A

NASA-TLX Survey Data

Task No.	**Avg. Mental Demand (%)**	**Avg. Stress Level (%)**
1	21.56	17.77
2	9.24	16.19
3	7.84	14.81
4	11.76	17.03
5	13.72	18.51
6	11.02	15.83
7	17.64	17.5
8	22.87	25.92
9	20.58	22.5
10	20.16	21.90
11	22.87	26.66
12	23.52	25.18
13	22.79	25.83
14	30.39	31.11
15	30.14	32.5
16	32.67	31.85
17	26.14	23.70
18	30.88	31.66
19	37.25	32.22
20	47.05	30.47
21	40	36
22	46.32	38.33
23	56.20	49.62
24	41.17	46.66
25	52.94	60

Participant Post Survey: Average percentage of self reported Mental Demand and Stress Level for all the 25 task which Based on NASA TLX

APPENDIX B

Experimental Tasks

Task1

Silently read the following paragraph:

"Consider the subtleness of the sea; how its most freaded creatures glide under water, unapparent for the most part, the treacherously hidden beneath the loveliest tints of azure. Consider also the devilish brilliance and beauty of many of its most remorseless tribes, as the dainty embellished shape of many species of sharks. Consider, once more, the universal cannibalism of the sea; all whose creatures prey upon each other, carrying on . eternal war since the world began. Please enter 135 into the input box below."

Task 2

```
public class task2{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      a = b;
      System.out.format("%d %d", a, b);
   }
}
```

Task 3

```
public class task3{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      b = a;
      System.out.format("%d %d", a, b);
   }
}
```

Task 4

```
public class task4{
   public static void main(String arg[]){
      int big = 10;
      int small = 20;
      big = small;
      System.out.format("%d %d", big, small);
   }
}
```

Task 5

```
public class task5{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      a = b;
      b = a;
      System.out.format("%d %d", a, b);
   }
}
```

Task 6

```
public class task6{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      b = a;
      a = b;
      System.out.format("%d %d", a, b);
   }
}
```

Task 7

```
public class task7{
   public static void main(String arg[]){
      int a = 10;
      int b = 20;
      int c = 30;
      a = b;
      b = c;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 8

```
public class task8{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      a = c;
      b = a;
      c = b;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 9

```
public class task9{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      c = b;
      b = a;
      a = c;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 10

```
public class task10{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      c = b;
      a = c;
      b = a;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 11

```
public class task11{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      b = a;
      c = b;
      a = c;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 12

```
public class task12{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      b = a;
      a = c;
      c = b;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 13

```
public class task13{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      a = c;
      c = b;
      b = a;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 14

```
public class task14{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      a = c;
      if ( a < b ){
         c = b;
      }
      b = a;
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 15

```
public class task15{
   public static void main(String arg[]){
      int a = 5;
      int b = 3;
      int c = 7;
      b = a;
      c = b;
      if ( c > a ){
         a = c;
      }
      System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 16

```
public class task16{
   public static void main(String arg[]){
   int a = 5;
   int b = 3;
   int c = 7;
   if ( a >= b ){
      b = a;
   }
   a = c;
   c = b;
   System.out.format("%d %d %d", a, b, c);
   }
}
```

Task 17

```
public class task17{
   public static void main(String arg[]){
   int array[] = {5, 3, 7};
   for (int i = 0; i < 3; i++){
      array[i] = 1;
   }
   System.out.format("%d %d %d", array[0],
         array[1], array[2]);
   }
}
```

Task 18

```
public class task18{
   public static void main(String arg[]){
      int array[] = {5, 3, 7, 4, 2, 6};
      int output = 0;
      for (int i = 0; i < 6; i++){
         output = output + array[i];
      }
      System.out.format("%d", output);
   }
}
```

Task 19

```
public class task19{

   public static void main(String arg[]){

      int array[] = {5, 3, 7, 4, 2, 6};

      int output = 0;

      for (int i = 5; i > 0; i--){

      if (array[i] % 2 == 0 )

         output = output + array[i];

      }

      System.out.format("%d", output);

   }

}
```

Task 20

```
public class task20{

   public static void main(String arg[]){

      int array[] = {5, 3, 7};

      int output = 0;

      for (int i = 0; i < 3; i++){

         for (int j = 0; j < 3; j++){

            output = output + array[j];

         }

      }

      System.out.format("%d", output);

   }

}
```

Task 21

```
public class task21{

   public static void main(String arg[]){

      int array[] = {5, 3, 7};

      int output = 0;

      for (int i = 0; i < 3; i++){

         output = output * i;

         for (int j = 0; j < 3; j++){

            output = output + array[j];

         }

      }

      System.out.format("%d", output);

   }

}
```

Task 22

```
public class task22{

   public static void main(String arg[]){

      int array[] = {5, 3, 7};

      int output = 0;

      for (int i = 0; i < 3; i++){

         if (array[i] > 4){

            for (int j = 0; j < 3; j++){

               output = output + array[j];

            }

         }

      }

      System.out.format("%d", output);

   }

}
```

Task 23

```
public class task23{
   private static int foo(int a){
      return a*2;
   }
   public static void main(String arg[]){
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 3; i++){
         array[i] = foo(array[i]);
      }
      System.out.format("%d %d %d", array[0],
          array[1], array[2]);
   }
}
```

Task 24

```
public class task24{
   private static int foo(int a, int b){
      return a+b;
   }
   public static void main(String arg[]){
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 3; i++){
         array[i] = foo(array[(i+1)%3], array[(i+2)%3]);
      }
      System.out.format("%d %d %d", array[0],
           array[1], array[2]);
   }
}
```

Task 25

```
public class task25{
   private static void foo(int[] a, int b, int c){
      int d;
      d = a[b];
      a[b] = a[c];
      a[c] = d;
   }
   public static void main(String arg[]){
      int array[] = {5, 3, 7};
      int output = 0;
      for (int i = 0; i < 4; i++){
         foo(array, i % 3, (i+1) % 3);
      }
   System.out.format("%d %d %d", array[0], array[1], array[2]);
   }
}
```