



Programa Interdisciplinar de Pós-Graduação em  
**Computação Aplicada**  
Mestrado Acadêmico

Matheus Segalotto

ARNI: an EEG-Based Model to Measure Program Comprehension

São Leopoldo, 2018



Matheus Segalotto

**ARNI: AN EEG-BASED MODEL TO MEASURE PROGRAM COMPREHENSION**

A dissertation is presented as a partial requirement to obtain the Master's Degree from the Graduate Program in Applied Computing at the University of Vale do Rio dos Sinos.

Advisor:  
Prof. Dr. Kleinner Silva Farias de Oliveira

São Leopoldo  
2018

S454a Segalotto, Matheus.

ARNI: an EEG-Based Model to Measure Program Comprehension / Matheus Segalotto. — 2018.

167 f. : il. color ; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, 2018.

“Orientador: Prof. Dr. Kleinner Silva Farias de Oliveira.”

1. Eletroencefalograma. 2. Estilo cognitivo. 3. Software - Desenvolvimento. 4. Compreensão de programa. I. Título.

CDU 004.41



Matheus Segalotto

**ARNI: AN EEG-BASED MODEL TO MEASURE PROGRAM COMPREHENSION**

A dissertation is presented as a partial requirement to obtain the Master's Degree from the Graduate Program in Applied Computing at the University of Vale do Rio dos Sinos.

Approved on January 18th, 2018.

**BANCA EXAMINADORA**

Prof. Dr. Kleinner Silva Farias de Oliveira – UNISINOS

Prof. Dr. Toacy Cavalcante de Oliveira – UFRJ

Prof. Dr. Jorge Luis Victória Barbosa – UNISINOS

Prof. Dr. Kleinner Silva Farias de Oliveira (Advisor)

Seen and allowed to print  
São Leopoldo

Prof. Dr. Rodrigo da Rosa Righi  
Coordinator PPG in Applied Computing



To my parents, because without them this would not be possible.  
Also, my fiancée, who has supported me during the hard times.

*“Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work. And the only way to do great work is to love what you do.”*

— STEVEN PAUL JOBS



## ACKNOWLEDGEMENTS

The development of this work would not have been possible without help from countless people in my life. I mention some specific individuals in the text below, but this does not restrict my appreciation for all the people who have helped me during this process. Thank you very much for your assistance, I truly appreciate it.

First, I would like to thank God who has given me so much in life. Also, thank you to my parents, João Alberto Segalotto and Cristine Inge Segalotto, for always encouraging me to study, creating a fertile environment for me to harvest my future, and for giving me the ability to reach my personal and professional goals.

Furthermore, I would like to thank my fiancée, Victória Koch, for her patience, support, and help through the difficult times while I was walking down this path. I also thank my brother, Michel Segalotto, for helping me with household chores to allow me more time to study and also taking me to UNISINOS whenever it was required.

I would especially like to thank my advisor, Kleinner Silva Farias de Oliveira. He always challenged me to study, gave me opportunities to enhance my career and was always willing to help me. He gave me the opportunity to continue his brain area work and was patient and available during this work. He gave me advice and suggestions when I did not know how to proceed. I also thank him for all of his attention and for his friendship with me.

Besides that, I'm thankful for the friendships created during this walk, highlighting two colleagues, Lucian José Gonçalves and Vinicius Bischoff, who supported me in the accomplishment of this work, motivating me and giving suggestions so that my study could improve over and over. At the same, I would like to thank all of the teachers from PPGCA (Postgraduate Program in Applied Computing), who helped me with valuable ideas regarding what this study might contribute to the scientific areas. I would especially like to thank Dr. Sandro José Rigo, Jorge Luis Victoria Barbosa, and João Francisco Valiati.

Thanks to the CAPES-PROSUP program for the scholarship, because without it, I probably would not be able to carry out this graduate course in the strict sense. Thank you to UNISINOS for the teaching and research environment in which they provided support to this research and to the laboratories used, where I was able to complete the execution of the experiment and spend the nights working in the study rooms available.

Last but not least, I wish to thank my relatives – my uncle, aunt, grandmother, grandfather, and cousins – for their understanding about my inability to give them due attention because of my restricted time, and all my friends and co-workers who in some way helped contribute with useful questions leading to the expansion of my knowledge. I would also like to thank my friends at Cognera in Canada, who gave me helpful ideas. In particular, my manager, Brendon Bedeau, who believed that I could handle my work and study at the same time, and David Hovey for his help with the English language in this dissertation.



*“For the far future, I envision a situation-dependent cognitive model of a single developer, which observes the current mental state of a developer and gives him or her the optimal support for the current situation.”*  
(SIEGMUND, 2016)





## RESUMO

A compreensão de programa é um processo cognitivo realizado no cérebro dos desenvolvedores para entender o código-fonte. Este processo cognitivo pode ser influenciado por vários fatores, incluindo o nível de modularização do código-fonte e o nível de experiência dos desenvolvedores de software. A compreensão de programa é amplamente reconhecida como uma tarefa com problemas de erro e esforço. No entanto, pouco foi feito para medir o esforço cognitivo dos desenvolvedores para compreender o programa. Além disso, esses fatores influentes não são explorados no nível de esforço cognitivo na perspectiva dos desenvolvedores de software. Além disso, alguns modelos de cognição foram criados para detectar indicadores de atividade cerebral, bem como dispositivos de eletroencefalografia (EEG) para suportar essas detecções. Infelizmente, eles não são capazes de medir o esforço cognitivo. Este trabalho, portanto, propõe o ARNI, um modelo computacional baseado em EEG para medir a compreensão do programa. O modelo ARNI foi produzido com base em lacunas encontradas na literatura após um estudo de mapeamento sistemático (SMS), que analisou 1706 estudos, 12 dos quais foram escolhidos como estudos primários. Um experimento controlado com 35 desenvolvedores de software foi realizado para avaliar o modelo ARNI através de 350 cenários de compreensão de programa. Além disso, esse experimento também avaliou os efeitos da modularização e a experiência dos desenvolvedores no esforço cognitivo dos desenvolvedores. Os resultados obtidos sugerem que o modelo ARNI foi útil para medir o esforço cognitivo. O experimento controlado revelou que a compreensão do código fonte não modular exigia menos esforço temporal (34,11%) e produziu uma taxa de compreensão mais alta (33,65%) do que o código fonte modular. As principais contribuições são: (1) a execução de SMS no contexto estudado; (2) um modelo computacional para medir a compreensão do programa para medir o código-fonte; (3) conhecimento empírico sobre os efeitos da modularização no esforço cognitivo dos desenvolvedores. Finalmente, este trabalho pode ser visto como um primeiro passo para uma agenda ambiciosa na área de compreensão de programa.

**Palavras-chave:** Eletroencefalograma. Estilo cognitivo. Software - Desenvolvimento. Compreensão de programa.



## ABSTRACT

Program comprehension is a cognitive process performed in the developers' brain to understand source code. This cognitive process may be influenced by several factors, including the modularization level of source code and the experience level of software developers. The program comprehension is widely recognized as an error-prone and effort-consuming task. However, little has been done to measure developers' cognitive effort to comprehend program. In addition, such influential factors are not explored at the cognitive effort level from the perspective of software developers. Additionally, some cognition models have been created to detect brain-activity indicators as well as wearable Electroencephalography (EEG) devices to support these detections. Unfortunately, they are not able to measure the cognitive effort. This work, therefore, proposes the ARNI, an EEG-Based computational model to measure program comprehension. The ARNI model was produced based on gaps found in the literature after a systematic mapping study (SMS), which reviewed 1706 studies, 12 of which were chosen as primary studies. A controlled experiment with 35 software developers was performed to evaluate the ARNI model through 350 scenarios of program comprehension. Moreover, this experiment also evaluated the effects of modularization and developers' experience on the developers' cognitive effort. The obtained results suggest that the ARNI model was useful to measure cognitive effort. The controlled experiment revealed that the comprehension of non-modular source code required less temporal effort (34.11%) and produced a higher correct comprehension rate (33.65%) than modular source code. The main contributions are: (1) the execution of SMS in the context studied; (2) a computational model to measure program comprehension to measure source code; (3) empirical knowledge about the effects of modularization on the developers' cognitive effort. Finally, this work can be seen as a first step for an ambitious agenda in the area of program comprehension.

**Keywords:** Electroencephalogram. Cognitive indicators. Experimental studies. Program comprehension.



## LIST OF FIGURES

Figure 1	Process to measure program comprehension. . . . .	33
Figure 2	Phases employed in this research. . . . .	36
Figure 3	Representation of factors associated with program comprehension. . . . .	38
Figure 4	Representation of developer's perception. . . . .	39
Figure 5	Integrated comprehension meta-model for program understanding. . . . .	41
Figure 6	An inception to a theory of software evolution processes. . . . .	43
Figure 7	The structured interactions between individual and physiological device. . .	44
Figure 8	Psychophysiological process in a software development context. . . . .	45
Figure 9	Overview of the process to extract biometric data and produce measurements. .	46
Figure 10	Activation pattern from a developers brain while comprehending code. . . .	47
Figure 11	EEG device and the electrodes location by international 10-20 system. . . .	49
Figure 12	Sample of EEG waves grouped by type over time. . . . .	51
Figure 13	An example of the cognitive process and working memory as a flow. . . . .	52
Figure 14	Alpha desynchronization during cognitive load activity. . . . .	53
Figure 15	Summary of the brainwave process from a dataset. . . . .	53
Figure 16	Form used for data extraction. . . . .	62
Figure 17	Phases for the filtering process in the systematic mapping review. . . . .	64
Figure 18	RQ6: Primary studies distribution by year. . . . .	69
Figure 19	Combining RQ2, RQ3, and Year. . . . .	73
Figure 20	Overview of the model processes. . . . .	79
Figure 21	Process of CognitiveEffort technique. . . . .	84
Figure 22	Calculation example of CognitiveEffort. . . . .	85
Figure 23	Developer overview of the execution process. . . . .	86
Figure 24	The methodology "4+1" view architecture. . . . .	87
Figure 25	Features diagram with components of ARNI model. . . . .	88
Figure 26	Process of ARNI model for program comprehension. . . . .	89
Figure 27	Component diagram of ARNI model. . . . .	90
Figure 28	Layer of ARNI model. . . . .	91
Figure 29	ARNI model prototype. . . . .	92
Figure 30	Task 5 - MT5 - modular scenario. . . . .	101
Figure 31	Proposed experimental process. . . . .	103
Figure 32	Components for the experiment. . . . .	107
Figure 33	Participant conducting the experiment. . . . .	109
Figure 34	Total temporal effort per task in the controlled experiment. . . . .	112
Figure 35	Answers and temporal effort per task. . . . .	113
Figure 36	Responses to the questions during the experimental phase. . . . .	115
Figure 37	Cognitive effort in the controlled experiment. . . . .	117
Figure 38	Non-modular Results grouped by experience . . . . .	120
Figure 39	Modular Results grouped by experience . . . . .	121
Figure 40	Scalp comparison using group's data from the experiment. . . . .	122
Figure 41	Frequency behavior in the non-modular group during a scenario. . . . .	122
Figure 42	Post experiment qualitative questionnaire using Likert scale. . . . .	124
Figure 43	Questionnaire - Page I. . . . .	153

Figure 44 Questionnaire - Page II. . . . . 154  
Figure 45 Questionnaire (After the Experiment). . . . . 155  
Figure 46 Non-Modularized - Question 1. . . . . 157  
Figure 47 Non-Modularized - Question 2. . . . . 157  
Figure 48 Non-Modularized - Question 3. . . . . 158  
Figure 49 Non-Modularized - Question 4. . . . . 158  
Figure 50 Non-Modularized - Question 5. . . . . 159  
Figure 51 Modularized - Question 1. . . . . 161  
Figure 52 Modularized - Question 2. . . . . 161  
Figure 53 Modularized - Question 3. . . . . 162  
Figure 54 Modularized - Question 4. . . . . 162  
Figure 55 Modularized - Question 5. . . . . 163

## LIST OF TABLES

Table 1	Research questions for systematic mapping study. . . . .	57
Table 2	Search strings used to retrieve the candidate articles. . . . .	59
Table 3	Databases used to find candidate studies. . . . .	60
Table 4	The inclusion criteria used to filter the studies retrieved. . . . .	61
Table 5	The exclusion criteria used to filter the studies retrieved. . . . .	61
Table 6	Keywords used to recover information from the selected studies. . . . .	62
Table 7	RQ1: BCI device supported. . . . .	65
Table 8	RQ2: Research strategies per empirical method. . . . .	66
Table 9	RQ3: variables investigated. . . . .	67
Table 10	RQ4: software abstraction. . . . .	68
Table 11	RQ5: influential factors. . . . .	68
Table 12	Comparison of related works. . . . .	72
Table 13	Variables used in the controlled experiment. . . . .	99
Table 14	Scenarios of the controlled experiment. . . . .	100
Table 15	Age and characteristics of the subjects. . . . .	104
Table 16	Participants' degree of schooling. . . . .	104
Table 17	Demographic data of participants (n = 35). . . . .	105
Table 18	Skill data of participants. . . . .	106
Table 19	Time consumed during the experiment. . . . .	108
Table 20	Descriptive statistic for the results (n = 35). . . . .	110
Table 21	Results per task and group using mean values. . . . .	110
Table 22	Results of normality test. . . . .	111
Table 23	Temporal effort results of statistical tests. . . . .	114
Table 24	Correct comprehension rate results of statistical tests. . . . .	116
Table 25	Statistical results for Cognitive effort. . . . .	118
Table 26	Participant's comparison amongst novice and expert. . . . .	120
Table 27	Questions in the qualitative questionnaire. . . . .	123
Table 28	Quality Assessment description. . . . .	149





## LIST OF ALGORITHMS

Algorithm 1	Load EEG data generated and split data based on markers. . . . .	81
Algorithm 2	Noise detection and pre-processing actions. . . . .	82
Algorithm 3	Techniques proposed to measure program comprehension. . . . .	83



## LIST OF ABBREVIATIONS

802.11	Wireless LAN
AT	ActiveTwo
AU	actiCap/g.USBamp
bps	bits per second
BT	Bluetooth
dB	decibel
DC	Direct Current
ECoG	EleCtrocorticoGraphy
EEG	ElectroEncephaloGraphy
e.g.	for example
EC	Electro-Cap/g.USBamp
i.e.	in essence
Hz	hertz
L1A	lower-1 alpha
ms	millisecond
min	minimum
med	median
max	maximum
MSc	Master of Science
mV	millivolts
null	none
p	probability
PhD	Doctor of Philosophy
px	pixel
SD	standard deviation
SVN	subversion
Unisinos	Universidade do vale do rio dos sinos
vi	visual editor
Wi-Fi	wireless



## LIST OF ACRONYMS

ABNT	Associação Brasileira de Normas Técnicas
ACM	Association for Computing Machinery
AD	Abstraction Degree
API	Application Program Interface
ARNI	Activity Related Neural Indicators
BA	Brodmann Areas
BCI	Brain-Computer Interfaces
BVP	Blood Volume Pulse
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CCR	Correct Comprehension Rate
CE	Cognitive Effort
CHASE	Cooperative and Human Aspects of Software Engineering
CLT	Cognitive Load Theory
COM	Component Object Model
CSV	Comma-Separated Values
DF	Degree of Freedom
DSL	Domain-Specific Language
EC	Exclusion Criteria
EDF	European Data Format
ERD	Event-Related Desynchronization
ERP	Event-Related Potential
fMRI	functional Magnetic Resonance Imaging
FFT	Fast Fourier Transform
GQM	Goal, Question, Metric
GSR	Galvanic Skin Response
HI	Halstead Length
HR	Heart Rate
HRV	Heart Rate Variability
IAF	Individual peak Alpha Frequency
IC	Inclusion Criteria
ICSE	International Conference on Software Engineering
ICSER	International Conference on Scientific and Engineering Research
IDE	Integrated Development Environment

IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IT	Information Technology
JS	Java Statement
LC	Level of Complexity
LOC	Lines of Code
M	Modular
MATLAB	Matrix Laboratory
MDD	McCabe Cyclomatic Complexity
MDD	Model-Driven Development
MI	Maintainability Index
MIT	Massachusetts Institute of Technology
MRI	Magnetic Resonance Imaging
NM	Non-Modular
OS	Operational System
PIO	Population, Intervention, and Outcome
PCE	Program Comprehension Effort
PPGCA	Postgraduate Program in Applied Computing
PnP	Plug-and-Play
PSD	Power Spectral Density
QA	Quality Assessment
RR	Respiratory Rate
RQ	Research Question
SMS	Systematic Mapping Study
SE	Software Engineering
SO	Specific Objectives
SS	Search Strings
SQL	Structured Query Language
SVM	Support Vector Machine
TE	Temporal Effort
UI	User Interface
UML	Unified Modeling Language
USB	Universal Serial Bus
VCS	Version Control System

## LIST OF SYMBOLS

$\leftarrow$	Attribution
$\alpha$	Alpha
$\in$	Belongs
$\beta$	Beta
$\emptyset$	Empty set
$=$	Equality
$\exists$	Exists
$\forall$	For All
$\gamma$	Gamma
$>$	Greater
$\leq$	Greater equal
$<$	Less
$\geq$	Less equal
$\{$	Left Curly Bracket
$\varphi$	Phi
$?$	Question mark
$\}$	Right Curly Bracket
$\Sigma$	Sum
$;$	Semicolon
$\theta$	Theta
$W$	Weight





## CONTENTS

<b>1 INTRODUCTION</b>	<b>31</b>
1.1 Motivation	31
1.2 Problem Statement	32
1.3 Research Question	34
1.4 Objectives	35
1.5 Methodology	36
1.6 Outline	36
<b>2 BACKGROUND</b>	<b>37</b>
2.1 Program Comprehension	37
2.1.1 Measuring Understanding	39
2.1.2 Systematizing Comprehension	40
2.1.3 Software Evolution	42
2.2 Psychophysiological Indicators	44
2.2.1 Biometric Data	45
2.2.2 Data Analysis	46
2.2.3 Developers Brain	47
2.3 Electroencephalography	48
2.3.1 Waveforms	50
2.3.2 Cognitive Load	52
<b>3 RELATED WORK</b>	<b>55</b>
3.1 Research Plan	55
3.1.1 Research Strategy	56
3.1.2 Data Extraction	58
3.1.3 Inclusion and Exclusion Criteria	60
3.2 Study Filtering	63
3.3 SMS Results	65
3.3.1 Comparative Studies	70
3.4 Research Opportunities	72
3.4.1 Cognitive Measurement	73
3.4.2 Measuring Comprehension	75
3.5 Limitations of the Evaluation	76
<b>4 ACTIVITY RELATED NEURAL INDICATORS (ARNI)</b>	<b>77</b>
4.1 Requirements	77
4.2 Concepts	79
4.3 Algorithms	81
4.4 Cognitive Effort Technique	83
4.5 Architecture	85
4.5.1 Features Model	87
4.5.2 Activity Diagram	88
4.5.3 Component Diagram	90
4.5.4 Multilayer Architecture	91
4.6 Aspects of Implementation	91
4.6.1 Technologies	93

<b>5 EVALUATION AND RESULTS</b>	<b>95</b>
<b>5.1 Experimental Drawing</b>	95
5.1.1 Objectives	96
5.1.2 Hypotheses	97
5.1.3 Variables	98
5.1.4 Scenarios	99
<b>5.2 Experiment Process</b>	102
5.2.1 Selection of Participants	103
5.2.2 Material	107
5.2.3 Execution	108
<b>5.3 Results</b>	109
5.3.1 Hypothesis 1: Temporal Effort	112
5.3.2 Hypothesis 2: Correctness	114
5.3.3 Hypothesis 3: Cognitive Effort	117
<b>5.4 Discussion</b>	119
5.4.1 Novice and Experts	119
5.4.2 Brain Activity	121
5.4.3 Experiment Retrospective	123
<b>5.5 Threats to Validity</b>	125
5.5.1 Construct	125
5.5.2 Statistical	126
5.5.3 Internal	126
5.5.4 External	127
<b>6 CONCLUSION</b>	<b>129</b>
<b>6.1 Contributions</b>	129
<b>6.2 Limitations of the Study</b>	130
<b>6.3 Future Works</b>	131
<b>REFERENCES</b>	<b>133</b>
<b>APPENDIX A – LIST OF PRIMARY STUDIES</b>	<b>147</b>
<b>APPENDIX B – QUALITY ASSESSMENT</b>	<b>149</b>
<b>APPENDIX C – STRING SEARCH ADAPTED BY ELECTRONIC DATABASE</b>	<b>151</b>
<b>APPENDIX D – QUESTIONNAIRE OF CHARACTERISTICS</b>	<b>153</b>
<b>APPENDIX E – QUESTIONNAIRE OF IMPRESSION</b>	<b>155</b>
<b>APPENDIX F – NON-MODULARIZED QUESTIONS</b>	<b>157</b>
<b>APPENDIX G – MODULARIZED QUESTIONS</b>	<b>161</b>
<b>ANNEX A – FREE AND EXCLUDED CONSENT TERM</b>	<b>165</b>
<b>ANNEX B – CONSENT LETTER</b>	<b>167</b>

## 1 INTRODUCTION

Program comprehension is a relevant cognitive process in software development. Developers may sometimes spend an enormous amount of time understanding a small fragment of all the source code (SIEGMUND, 2016). Additionally, up to 50% of the whole time in software maintenance may be spent reviewing and understanding source code (BORSTLER; PAECH, 2016; NGUYEN; BOEHM; DANPHITSANUPHAN, 2011).

Over the last few years, new cognition models have been produced to provide detectable indicators of brain activity using brain-computer interface (BCI) devices such as the electroencephalogram (EEG) (CRK; KLUTHE; STEFIK, 2015a). Currently, an understanding of how and why these improvements can help developers is severely lacking. Even worse, academia and the industry have neglected to produce a broad meta-analysis of the current literature addressing the use of BCI to reveal the required cognitive load to comprehend a program.

This chapter aims to present the main purposes and structure of this study. For that reason, Section 1.1 introduces the motivation for this study, giving a panoramic view of the motivation of this research. Section 1.2 describes the problem presented and discussed in this study. After that, Section 1.3 highlights the research question (RQ) that this study will answer. The objectives of this study are presented in Section 1.4. Section 1.5 describes the methodology used to reach the goal of this research. Section 1.6 gives a summary of the structure of this study.

### 1.1 Motivation

In the last few decades, software engineering (SE) has been searching for new ways to reduce the total time to perform software maintenance (NGUYEN, 2010; FOSTER, 1993; YIP; LAM, 1994). Recently, substantial advancements have been achieved in different areas of software development, such as integrated development environments (IDEs), developer dashboards for supporting qualitative analytics, like SonarQube<sup>1</sup>, and more instinctive programming languages, such as Swift<sup>2</sup> (BAYSAL; HOLMES; GODFREY, 2013).

The source code is becoming more complex every day, and to support this, many processes and techniques have been developed (MAYRHAUSER; VANS, 1998). Extensive documentation can help during maintenance but also brings with it great trade-off for long-term documentation since it can deteriorate rapidly (ARISHOLM et al., 2006; LETHBRIDGE; SINGER; FORWARD, 2003; SPINELLIS, 2010). In the agile/lean development practices, the code must be easily understood, especially for developers applying practices or coding guidelines that improve program comprehension (PETERSON, 2010; MAYRHAUSER; VANS, 1998; KELLERHER; PAUSCH, 2005).

Currently, three approaches can be used to measure program comprehension (SIEGMUND,

---

<sup>1</sup>SonarQube: <http://www.sonarqube.org/>

<sup>2</sup>Swift: <https://swift.org/>

2016). The first approach is the think-aloud protocol, which asks the participants to verbalize their thoughts during the interpretation process; it is also known as the oldest method (WUNDT, 1895). The second is the memorization method, which tests the performance that developers can subsequently recall a piece of source code (SHNEIDERMAN, 1976). The third is the comprehension task, which requires the participant to fill in an empty space by properly executing a piece of source code (SOLOWAY; EHRLICH, 1984).

Regardless of the current advances in software comprehension, it is still a very difficult activity to measure. In (CRK; KLUTHE; STEFIK, 2015b), the authors measure the experience of some developer groups using a non-invasive electroencephalogram. This technique is based on an event-related desynchronization (ERD). This method captures the relaxed mental state of a person and compares it with the active mental state. The difference between the states is estimated. In (FRITZ; MÜLLER, 2016), multiple biometric devices are used to measure the developers' psychophysical behavior: an EEG, a heart rate (HR), and also a galvanic skin response (GSR). The technique uses machine learning to predict when a software developer may have a productivity issue in one of his tasks. All these methods were developed to support a developers' daily work, giving metrics that were not previously developed.

In the absence of any automatic measure, for the quality of the source code, the goal of this study is to build a model and technique to measure program comprehension. This focuses on measuring the effort required to developers to understand a fragment of the source code. To achieve this, the study will use a variation of effort from relaxing mode into a program understanding mode. The difference between these two states is calculated, generating a method to evaluate a method to measure the cognitive effort generated by a specific developer to understand a source code.

In general, quantifying the program comprehension has been difficult as all techniques use the output generated by a developer instead of measuring the developer themselves. For instance, there is no simple way currently to compare if one source code snippet is easier or harder than another source code snippet based on the developer's background. Also, how do we know which project patterns bring more benefits than side-effects? Consequently, an in-depth understanding of the state-of-the-art works remains limited and inconclusive. In addition, modularization is defined in this study as the division of a code into smaller blocks. Code division is mainly used to facilitate maintenance and increase reuse in programming languages.

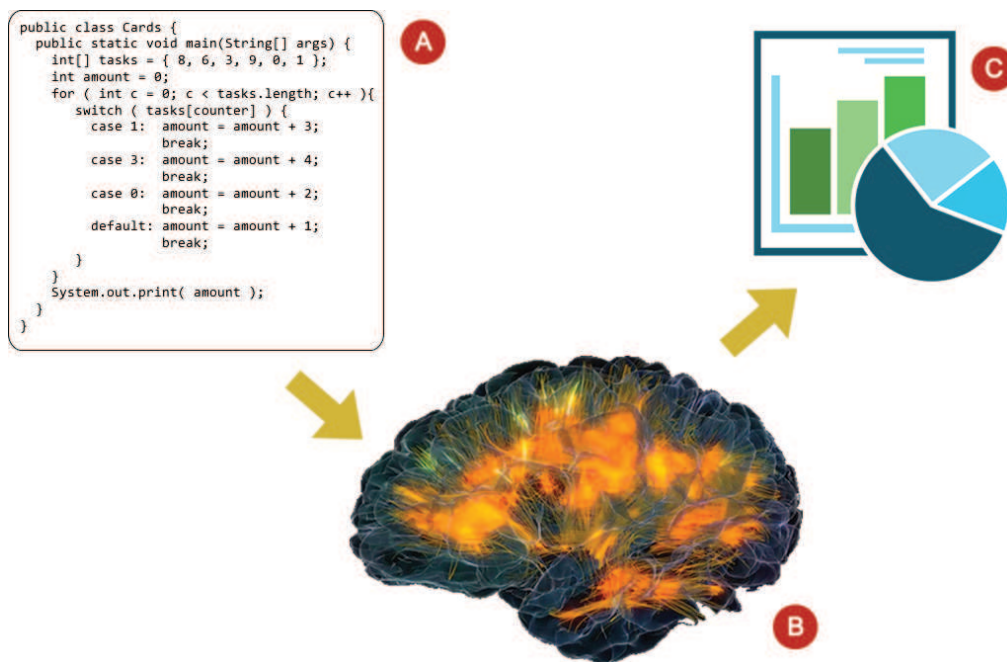
## 1.2 Problem Statement

The problematic focus is measuring the program comprehension using an electroencephalogram. The cognitive process of program comprehension happens when developers brain reads and interprets a source code snippet (MAYRHAUSER; VANS, 1995). The brain activity can be measured by the number of times that the neurons are enabled and disabled, and this action generates an electromagnetic change which can be captured by an EEG device. Lastly, the level

of effort to understand the source code is defined by the variation from a relaxed mental state to a focused mental state, which will establish the degree of effort.

Figure 1 displays the process to measure the program comprehension when a developer is analyzing a snippet of source code. This cognitive process generates electric pulses when a neuron is activated, turning on and off in the brain. After capturing this through sensitive electrodes, the electronic sign is translated into brain waves. These raw data is then processed through wave pattern analysis and waveforms classification based on where are the brain this occurred. This problem materializes when processing the raw EEG and finding a definable pattern which makes this task hard to accomplish. This study will use, as an example of brain pattern, the modularization method in the source code. Nevertheless, it does not limit this study to apply to additional problems in the source code.

Figure 1 – Process to measure program comprehension.



Source: Created by the author

The software developed languages have been becoming more familiar the human language since its born. Figure 1A displays a snippet of source code in JAVA program language which is considered one of most popular software languages in both the academia and industry (THUNG et al., 2013). However, a miscellaneous set of languages for software development exists and each of them can have their own design patterns, including JAVA (SIEGMUND, 2016). Figure 1B illustrates the synapses in developers brain, in which they are part of the cognitive process. Synapse is a pulse that is transmitted from a neuron to another one generating an electric charge. When multiple synapses happen in the same brain area, it generates an electric field that can be captured by an non-invasive electroencephalogram (NUNEZ; SRINIVASAN, 2006).

Going one step further, Figure 1C presents the results of the brain waves processed through a model. Measuring how a specific developer understands and learns may help identify which software languages and design patterns are better when compared with one another. Further, a metric is missing to measure brain activity in the field of software development.

According to (BORSTLER; PAECH, 2016) and (SIEGMUND, 2016), the measure of program comprehension in software development is an open problem and hard to measure. Regarding the problems characterized above, this study investigates the three Problems (P) in demand, which are described below:

- **P-1: The absence of a wide-angle view of the state of the art.** In the last few years, several studies have analyzed the human factor in the computer software area, especially using the non-invasive electroencephalogram. Little is known about the types of BCI devices and variables manipulated, and about the kind of research strategies and criteria of software abstraction the studies have used. Hence, it becomes relevant to explore the gaps and trends in academia, and also highlight the opportunities remaining.
- **P-2: A lack of awareness of the techniques to measure program comprehension using an electroencephalogram.** The main difficulty resides in comparing if a new design pattern, or even a new software language, is improving our ecosystem. As a result, we frequently use the output generated as an input for a metric, not considering how much of the developer's effort is used to complete a task. Furthermore, a technical component is missing to measure the brain effort spent to perform a task, which is an important resource in the process.
- **P-3: Absence of empirical knowledge about the effects of modularization techniques in the developer's brain.** The modularization of software is widespread in several software languages. Hence, academia and industry started adopting this approach to focus on reducing costs and increasing productivity (DASHOFY; HOEK; TAYLOR, 2005). Nothing has been done to evaluate the degrees of modularization. Also, the comprehension of its trade-off from the developer's brain effort perceptive and identification when higher or lower degree of modularity is applied.

### 1.3 Research Question

As a result of the problem statement, it is necessary to determine a method that can quantify the mental effort generated by a developer. Additionally, this is a missing metric that has the potential to evaluate source code variations and also improve the personal abilities of each developer's characteristics. The current program comprehension methods are not able to represent mental effort, and this new method might represent a breakthrough in the academia and software industry. Therefore, the main research question for this study is presented as follows:



**General Research Question:** How do we measure the developer's mental effort during the reading and understanding of source code?

To answer this general research question, three specific research questions were developed. These questions are outlined as follows:

- **RQ-1:** What is the state-of-the-art regarding program comprehension in software development using BCI devices?
- **RQ-2:** How to estimate the cognitive effort collected by BCI devices to measure the understanding of source code in developers?
- **RQ-3:** How is the impact of modularization on the cognitive effort during the comprehension process?

#### 1.4 Objectives

Since Section 1.3 has presented the problem statement and emphasized the research questions, this section presents the objectives explored throughout this study. Further, the main objective for this study is described below:

**General Objective:** Propose a computational model for measuring developer's brain activities during program comprehension, and evaluate the effects of source code modularization on the developer's cognitive effort.

Hence, to answer the research questions, we identified three specific objectives (SO), derived from the main objective, and they are:

- **SO-1: Conduct a systematic mapping study of the literature in the program comprehension area.** This objective focuses on resolving the lack of a wide-angle view of the state of the art in the program comprehension area using an electroencephalogram. It focuses on collecting what was done, generates a big picture, and elucidates opportunities on the basis of the results.
- **SO-2: Propose and implement a model and technique to measure program comprehension using an EEG device.** Employing the literature as a foundation, this section aims to create a method that will translate electromagnetic impositions from specific parts of the brain and measure the developer's level of effort.
- **SO-3: Compose empirical knowledge on the effects of the modularization technique in the developer's brain.** This technique aims to generate empirical knowledge using the developed technique to measure brain effort. Currently, we do not know which level of modularization brings benefits and which becomes an issue on the basis of mental effort.

## 1.5 Methodology

This section details the study methodology adopted to reach the SOs described in Section 1.4. Additionally, each chapter in this study further explores the issues in order to address the RQs from Section 1.3. Figure 2 illustrates the methodologies applied in this study.

Figure 2 – Phases employed in this research.



Source: Created by the author

The first phase of this study performed a literature review about program comprehension and BCI devices, using a systematic mapping review, to accomplish the first research question (PETERSEN; VAKKALANKA; KUZNIARZ, 2015a). This initial research has provided a big picture of what has been done, gathered information and also introduced new research opportunities. Besides, this step was crucial to learn the area better.

In the second phase, a controlled experiment was conducted by collecting the signal of brain waves from 35 developers during ten distinct questions divided into five progressive levels of complexity. Based on the brain waves stored, a new technique has been proposed to accomplish the second research question. Also, pilot tests were executed to understand brain patterns.

In the third and final phase, the technique developed was used to measure the level of effort that each developer employed in several levels of software abstraction. Further, empirical data was generated using the technique developed to measure program comprehension from a new perspective. After a deep analysis, this step accomplished the third research question.

## 1.6 Outline

This remaining study is organized as follows: Chapter 2 introduces the main concepts about program comprehension, neurophysiological indicators, and electroencephalography, terms whose comprehension is required to grasp our literature review. Chapter 3 presents the related work obtained through a detailed systematic mapping review and details a list with relevant research opportunities. Chapter 4 explains the model and strategies to measure comprehension, including the technique developed in this study. Chapter 5 discusses the results obtained, as well as the threats to the validity of this evaluation. Finally, Chapter 6 presents concluding remarks, study limitations, and the challenges for further research.



## 2 BACKGROUND

With the main objective to develop a technique to measure brain activity when a developer understands the source code, it is crucial to gather concepts and technologies to use along the way. This chapter aims at assembling knowledge used to create and support the development proposed in this study.

The action of reading source code and combining it with the existing knowledge succinctly explains how the program comprehension process works. For the purpose of better explaining this process, multiple mental models were developed to describe the mental process from a software developers perspective. The action of creating a high-level abstraction from a low-level detailed perspective is called chunking (MAYRHAUSER; VANS, 1995). Three of these models are currently used to represent the understanding process in the computer science area: top-down, bottom-up, and an integrated model (SIEGMUND, 2016).

Cognitive activity can be summarized by an enormous amount of neurons generating electric pulses in different places within humans brain. The relationship between the mental and physical response from the human body is where biometric data comes into the picture. The connection between mind and body is the definition of the psychophysiological area (FRISTON et al., 1997). With the evolution of the electronic sensors, especially from a hardware perspective, they are becoming smaller and more accurate over the last few years. The electroencephalograph area is less invasive which allow humans to translate the magnetic variation generated from human brain in quantitative values for software analysis (GALÁN et al., 2008). The theme psychophysiological indicators will be discussed with the purpose of relating electroencephalograph and program comprehension focusing on the software development area.

This chapter is organized into three sections, where each section represents a pillar from abstract thinking into an effort metric. Primarily, Section 2.1 introduces aspects about program comprehension. Existing methods to measure software comprehension are discussed by understanding models. Section 2.2 presents some important concepts about the relationship between mental and physical processes and the different ways that they can be measured. Lastly, Section 2.3 outlines electroencephalogram and the waveform applied in pattern analysis using the EEGlab as a tool to support it.

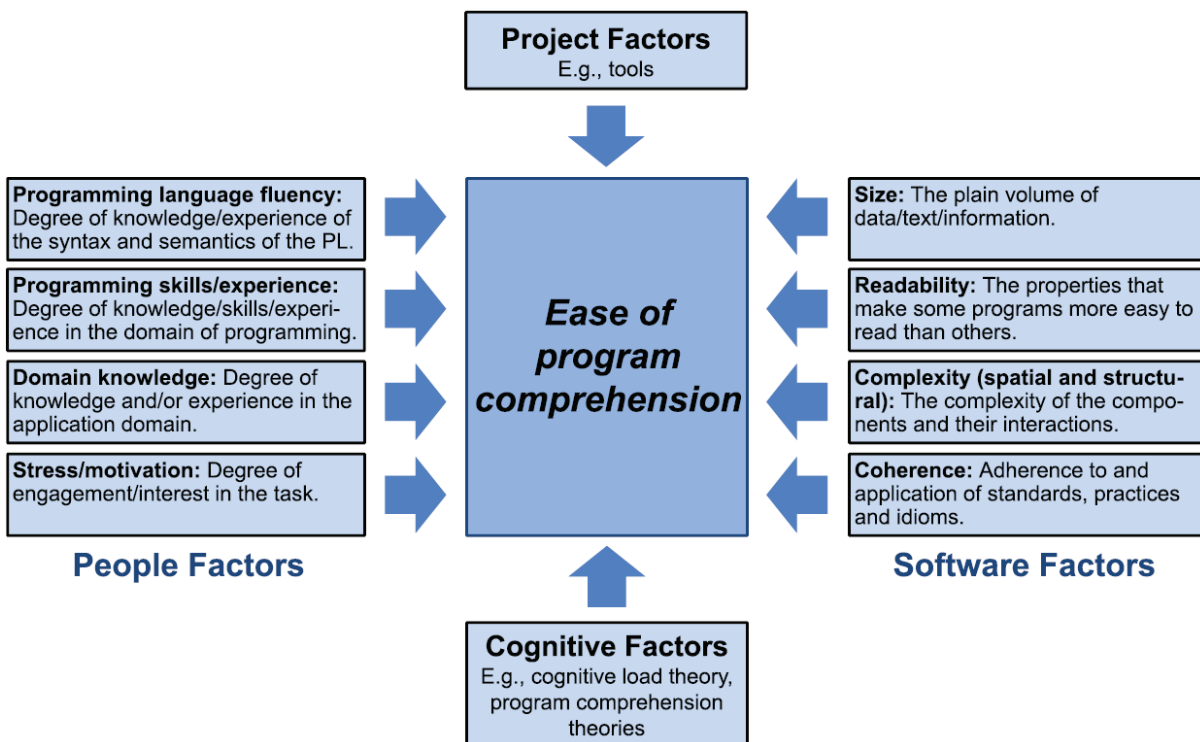
### 2.1 Program Comprehension

This section presents concepts about program comprehension and describes the relevance in software development of this cognitive process (SIEGMUND, 2016). The action of translating logic from a few keywords and having all of them make sense in our brain requires an enormous amount of effort (SIEGMUND; SCHUMANN, 2014). Source code written by different authors and using exceptional standards and terminologies can easily make a simple task complex to understand (RAJLICH, 2009).

Computer science is not the first area to deal with comprehension problems and undoubtedly will not be the last. Fields such as linguistics, mathematics, and artificial intelligence have to handle this issue through their own methodology (RAJLICH, 2009; KOZACZYNSKI; NING; ENGBERTS, 1992). During the lifetime of software, between 66 and 90 percent of the total expenses comes from software maintenance tasks (ERLIKH, 2000; YIP; LAM, 1994; FOSTER, 1993). Also, an experiment suggests that up to half of the total effort in a maintenance task is spent on program comprehension (NGUYEN, 2010; BOEHM, 2000).

Comprehension ease is divided into four groups of factors, and the list of classification factors for each is shown in Figure 3. Also, the people factor represents the background of developers while the software factor correlates characteristics from the source code. Cognitive factors represent mental activities that happen within the developer’s brain (STOREY, 2005).

Figure 3 – Representation of factors associated with program comprehension.



Source: Borstler e Paech (2016)

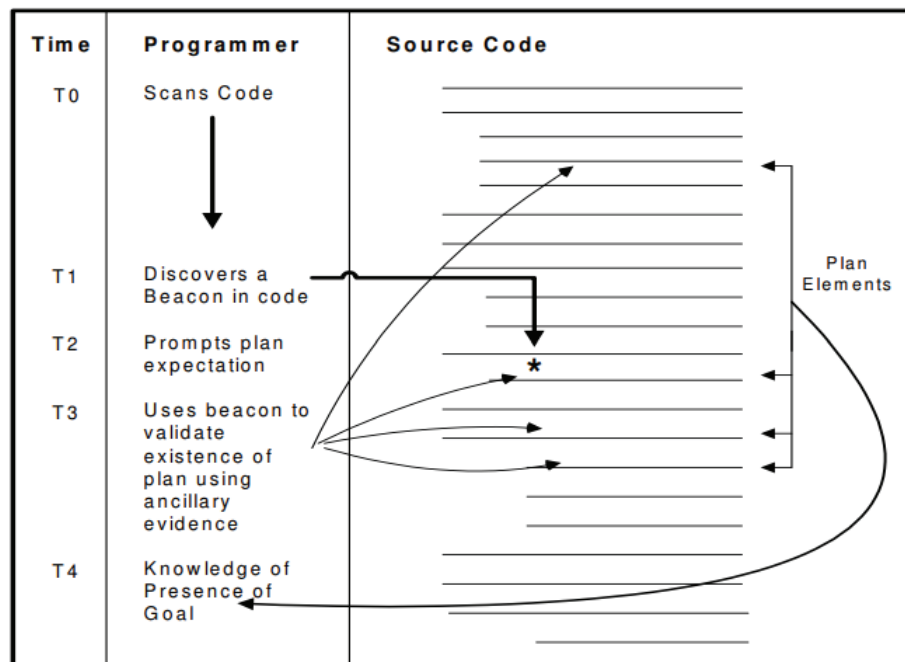
Over the last 30 years of research, little has been done to notably reduce the effort on comprehension tasks (MAYRHAUSER; VANS, 1998; IWASA, 2009; KLINGNER; TVERSKY; HANRAHAN, 2011). Numerous solutions were created attempting to solve this puzzle, though few evaluated the developer obligations sufficiently (SIEGMUND, 2016). Further, models of program comprehension attempt to describe the developer’s thinking process while absorbing the source code into three models (MAYRHAUSER; VANS, 1995).

### 2.1.1 Measuring Understanding

To measure how a developer understands source code in experiments, a limited amount of methods can be selected. Some of these approaches include think-aloud protocols, memorization, and comprehension tasks that have several implications, since human factor are not entirely considered (SHNEIDERMAN, 1976; SIEGMUND, 2016).

Figure 4 illustrates opportunistic understanding in a source code snippet which might differentiate due to human factor (O'BRIEN, 2003). Two main concepts are connected with the developer's perception. First, the programming plans are known as fragments of source code in software development. Second, the concept beacon is associated to recognize conventional structures that occur in the source code (BROOKS, 1983; SOLOWAY; EHRLICH, 1984).

Figure 4 – Representation of developer's perception.



Source: O'Brien (2003)

Primarily, for a developer to understand any type of source code is scanning it as shown on Figure 4 (T0). After that, Figure 4 (T1) illustrates the presence how beacons are noticed in the code while the programming plan is being created as displayed on Figure 4 (T2) (SOLOWAY; EHRLICH, 1984). After creating a programming plan, Figure 4 (T3) describes the goal to validate comprehension against the code (MAYRHAUSER; VANS, 1995; GILMORE; GREEN, 1988). As the final step, Figure 4 (T4) demonstrates the knowledge created in the process, and in this way, summarizing the complexity involved to measure program comprehension from a developer's perspective (O'BRIEN, 2003; DAVIES, 1990; RIST, 1987).

**Think-Aloud Protocol** Also known as introspection, this method which observes cognitive process has been used for over 140 years to measure subjective thought (WUNDT, 1895; FOX; ERICSSON; BEST, 2011). The experiment is typically recorded in audio and/or video format. The recorded information is then transcribed and used to analyze participants thoughts. It is commonly used to observe experiments involving the comprehension process (ERICSSON; SIMON, 1984). The most obvious problem with this technique is related to the effort needed (SIEGMUND, 2016). Some studies show it might bring a high level of imprecision as we do not have complete control of our brain activity (SCHOOLER, 2011; SHAFT; VESSEY, 1995).

**Memorization** This method focuses on remembering the subsequent recall of code based on the current line, which is easier if the developer understands the source code (SHNEIDERMAN, 1976). In this method, memorization is used to measure program comprehension and record the response time to evaluate the influences corresponding to the tasks (SHNEIDERMAN; BEN, 1980). During the task, developers should memorize and decide whether the next line is part of the study or not (PENNINGTON, 1987b) This method estimates a small portion of the tasks involved, which does not adequately represent the program understanding process (SOLOWAY; EHRLICH, 1984; SIEGMUND, 2016).

**Comprehension Task** This method explores the measurement through filling in the blanks in the source code (EHOW, 2014). Some experiments use empty spaces in the code of origin, such as words or even lines, for a developer to complete. Since there is not an option to choose from, this type of experiment requires more attention than the memorization technique (FRITZ et al., 2014; SOLOWAY; EHRLICH, 1984). Nevertheless, different operators or syntax can lead to distinct results which can happen because the developer needs to understand to fill in the field instead of understanding prior to filing the complete source code (SIEGMUND, 2016)

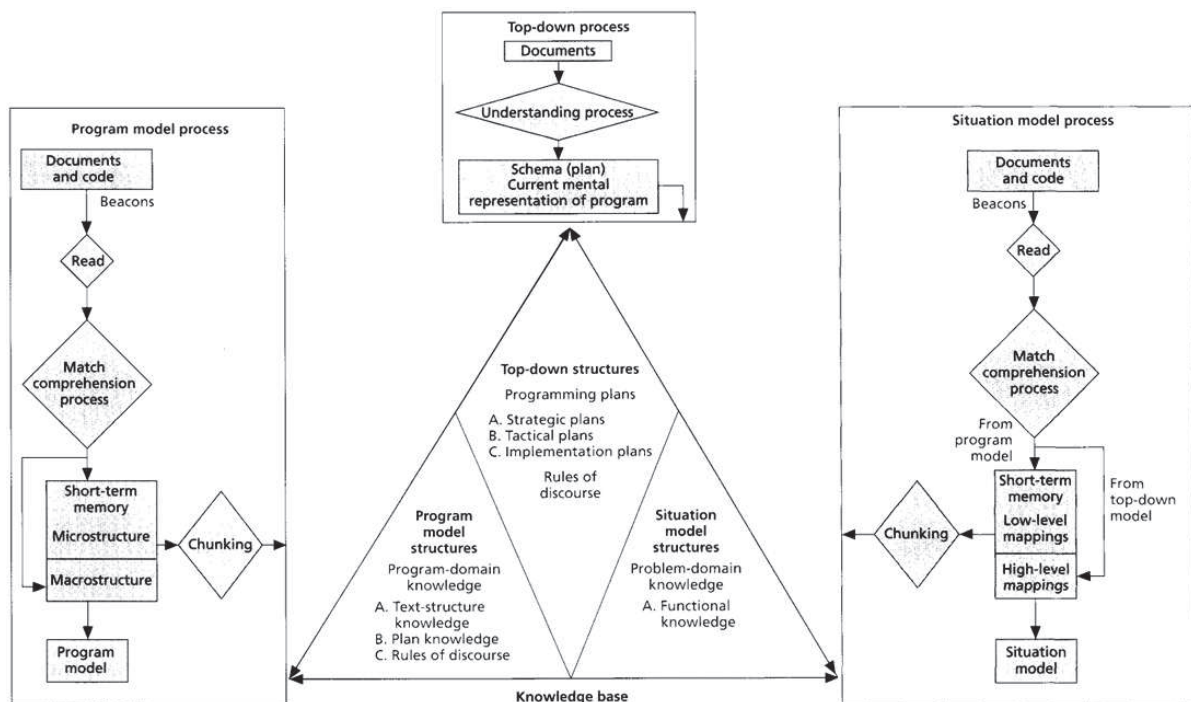
The available approaches in measuring program comprehension seem somewhat imprecise nowadays, especially with the variety of languages and tools available. A gap to evaluate the mental activity exists in academia while the experiment use the old methods such as task results (SIEGMUND, 2016). To better understand how a developer creates their understanding during the comprehension process, a few models have been set up to interpret the program comprehension action, which is discussed in the next subsection.

### 2.1.2 Systematizing Comprehension

In the process of software comprehension, there are three main models developed, which use to describe the mental activity: top-down, bottom-up and integrated meta-model (SIEGMUND, 2016). As a high-level explanation, the top-down model applies a developer's assumption about the code before digging into the actual source code. The bottom-up approach applies the reverse logic, and it starts directly inside the code, without knowing anything about the context or about a particular domain. Finally, the meta-model integrates the top-down and bottom-up models and enforces the top-down model to reduce mental effort (ROEHM et al., 2012).

Figure 5 is composed of three comprehension models, and their steps are listed inside each frame representing the understanding of the source code (KOENEMANN; ROBERTSON, 1991; MAYRHAUSER; VANS, 1995). The first model is the top-down understanding of Soloway, Adelson e Ehrlich (1988) and the second one is known as the bottom-up understanding of Pennington (1987b). Finally, the last mental model is the integrated model, and it is specifically used in large systems which switch between the first two models (PENNINGTON, 1987a; MAYRHAUSER; VANS, 1994).

Figure 5 – Integrated comprehension meta-model for program understanding.



Source: Mayrhauser e Vans (1995)

The program comprehension model comes with the grouping of three sub-models with the integrated model being the main one. A summary of each comprehension model illustrated in Figure 5 is listed below:

**Top-Down** It is commonly used when developers are familiar with the domain knowledge of a program. In this scenario, programmers hypothesize about system behavior, which allows that they find a faster solution with a small amount of information (MAYRHAUSER; VANS, 1994). Beacons are used to evaluate if the hypothesis created is a valid scenario or they misunderstood the program. If the hypothesis is denied, the developer will have to reformulate and test again. Sub-hypothesis can derive from the main hypothesis, helping in the understanding process (SOLOWAY; EHRLICH, 1984). Ignoring the details is considered a huge drawback, while developers only focus on high level aspects, sometimes speculations, making them ignore how the program works. (BROOKS, 1983).

**Bottom-Up** Diving into source code is the first step in circumstances where developers have inadequate or insufficient domain knowledge. Chunks are created after investigating several methods of a program (PENNINGTON, 1987b). In this scenario, higher-level abstractions, known as chunks, connect with other chunks to create into a larger chunk. After the examination of the source code, an abstraction is created, and the purpose of the program is then understood (SIEGMUND, 2016). Conversely, the bottoms up model requires notably more effort to completely understand the purpose of a program while it knows exactly how the code works and without making any assumption regarding it (SHNEIDERMAN, 1976).

**Integrated** The use of the two models above is known as an integrated model and is most used in software development (MAYRHAUSER; VANS, 1994). Combining the efficiency of top-down whenever a developer knows the domain knowledge together with the bottom-up model to resolve their difficulties in a code statement becomes a smart approach to decrease costs and enhance quality (SHAFT; VESSEY, 1995). Nevertheless, producing a program is becoming much more than just creating source code. The level of complexity and increased strategies to develop are formulating a gap of how we think while understanding a program (WARD, 1994; O'BRIEN, 2003).

Knowing how the mental model knowledge is relevant to understanding the developer's mental activity, sub-section 2.1.3 will explore how software has evolved over the last few decades and what is the role of program comprehension in the software environment and technology evolution.

### 2.1.3 Software Evolution

Machine code is the first programming language, and since then, it has been evolving from the combination of 1's and 0's used in the early days. To explain this evolution, the foundation of software development can be divided into two parts. The first is based on observation, hypotheses and assumptions from the real world and the second is the theoretical process of software evolution as shown in Figure 6.

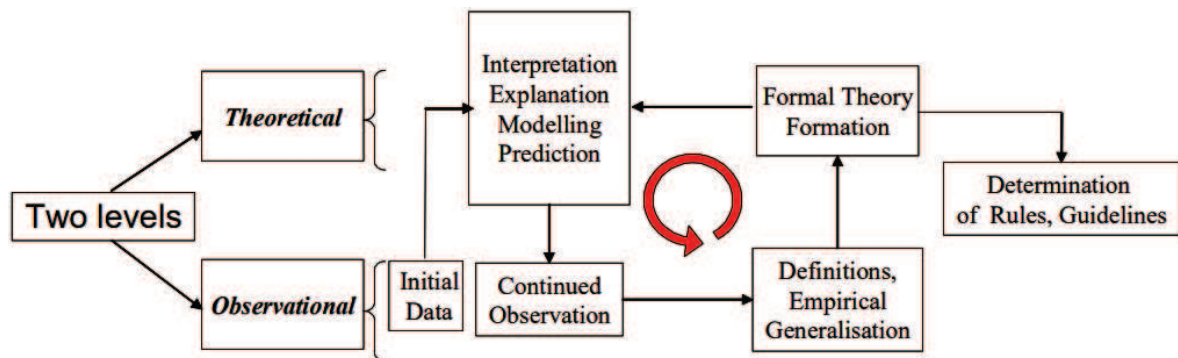
Since the creation of the assembly language, several higher-level programming languages were developed, such as Pascal, COBOL, JavaScript, JAVA and Swift (SALOMON, 1993). Each of them improves on how we use mnemonics and code structure to increase the understanding and better reflect the purpose of what tasks the program is going to be doing, especially in someone else's code (LEHMAN; RAMIL, 2003).

Application program interface (API) has been created to provide support for reusable implementation, reducing development time once the developer has a high-level knowledge how that API works. Domain-specific languages (DSLs) were developed for a specific domain that occasionally makes the code similar to human language, and consequently, easier to understand such as the structured query language (SQL) language (SIEGMUND, 2016).

At the beginning of software development, the developer only had a basic screen-oriented



Figure 6 – An inception to a theory of software evolution processes.



Source: Lehman e Ramil (2003)

text editor such as Emacs or vi to develop. Once the programs started becoming more complex, the standard editors were replaced by very powerful IDEs, also known as frameworks (NEWMAN, 1982).

Extensions, also known as plug-ins, allow customizations such as indentation spacing and color-coding styles. Also, auto-completion of words as a user was still typing and the debug process to find and resolve defects are a small portion of functionalities that IDEs brought to enhance the easy comprehension in the development environment (MIARA et al., 1983; RAMBALLY, 1986). Researchers in the mid-90s helped us to understand how code layout can significantly affect program comprehension in scenarios like the scope of a loop, variable declaration and the use of APIs (SIEGMUND, 2016).

Rules of programming can be defined by establishing protocols for development, like algorithm implementations as well as coding standards (STOREY, 2005). The primary purpose concerns program comprehension which becomes even more important for large organizations of software development (SOLOWAY; EHRLICH, 1984).

Modularity is a very familiar design pattern used in program maintainability nowadays. It allows the reduction of effort by a developer by reusing existing source code. At the same time, a module should be understood by itself, having no dependences with others modules. This simplicity reduces the effort of understanding and it creates less chunks in a developers brain (ESTEVAJ.C.; REYNOLDSR.G., 1991; MEYER, 1997). Furthermore, the literature does not stipulate any specific degree of modularity to be used in a source code as well as its side effect such as program comprehension.

The history of software development shows the evolution in program comprehension areas as well as in the software environment itself. Design patterns and coding standards, such as modularity and indentation respectively, illustrate the need for obtaining original and helpful ways to improve understanding from a developer's perspective. Only a few methods attempt to quantify the benefits from a proposed solution related to easy comprehension. That might

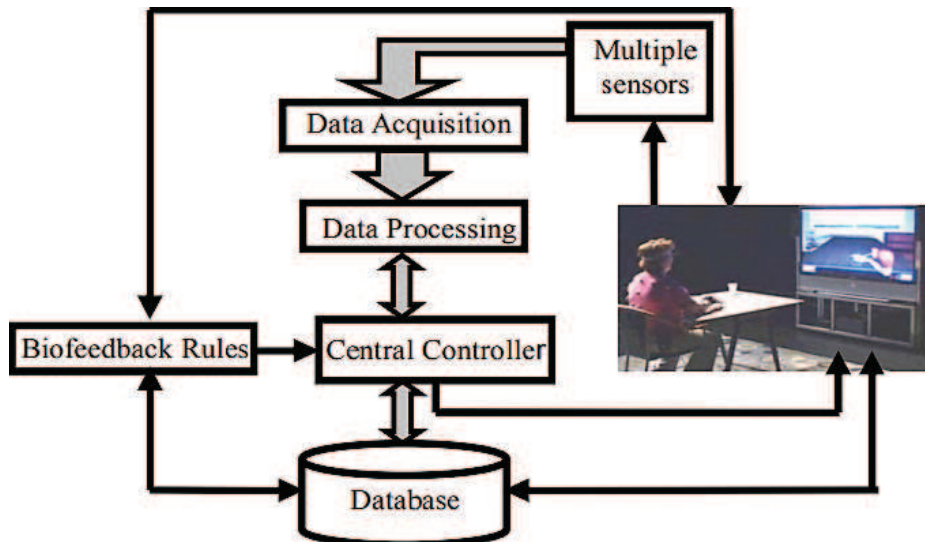
explain the gap created in academia about this field. Section 2.2 aims to comprehend rational behavior associated with physiological reactions and how the electrical impulses can be translated with the help of biological devices such as BCI machines.

## 2.2 Psychophysiological Indicators

The field that relates mental states and their physiological reactions is known as the psychophysiology process (ANDREASSI, 2000). In the last few years, substantial studies have explored techniques to extract behavior from a person, such as high or low cognitive load, arousal and valence, and also cognitive and emotional states (RICHTER et al., 1998; SJØGAARD; LUNDBERG; KADEFORS, 2000; LIU et al., 2016; BALDUCCI; GRANA; CUCCHIARA, 2017). Additionally, electro-dermal activity using EEG has been successful predicting human behavior, including some software developer activities (SWAIN, 1996; WILSON, 2002; MURUGAPPAN et al., 2007; FRITZ; MÜLLER, 2016).

To provide a relation between mental actions and the electric pattern is not an easy task. Figure 7 illustrates, at a high level, how this behavior works between a user and a software application. Multi-sensors are used to monitor biofeedback and apply data acquisition and processing multiple times from a user while biofeedback rules are triggered when a specific pattern occurs in the central controller (ALLANSON; FAIRCLOUGH, 2004).

Figure 7 – The structured interactions between individual and physiological device.



Source: He Huang et al. (2005)

Several methods to measure biometric data have been raised, and each of them can be categorized by the way they extracts and transform data (SHARIT; SALVENDY, 2006). Collecting data is only the first step; preprocessing the data and then analyzing patterns usually requires

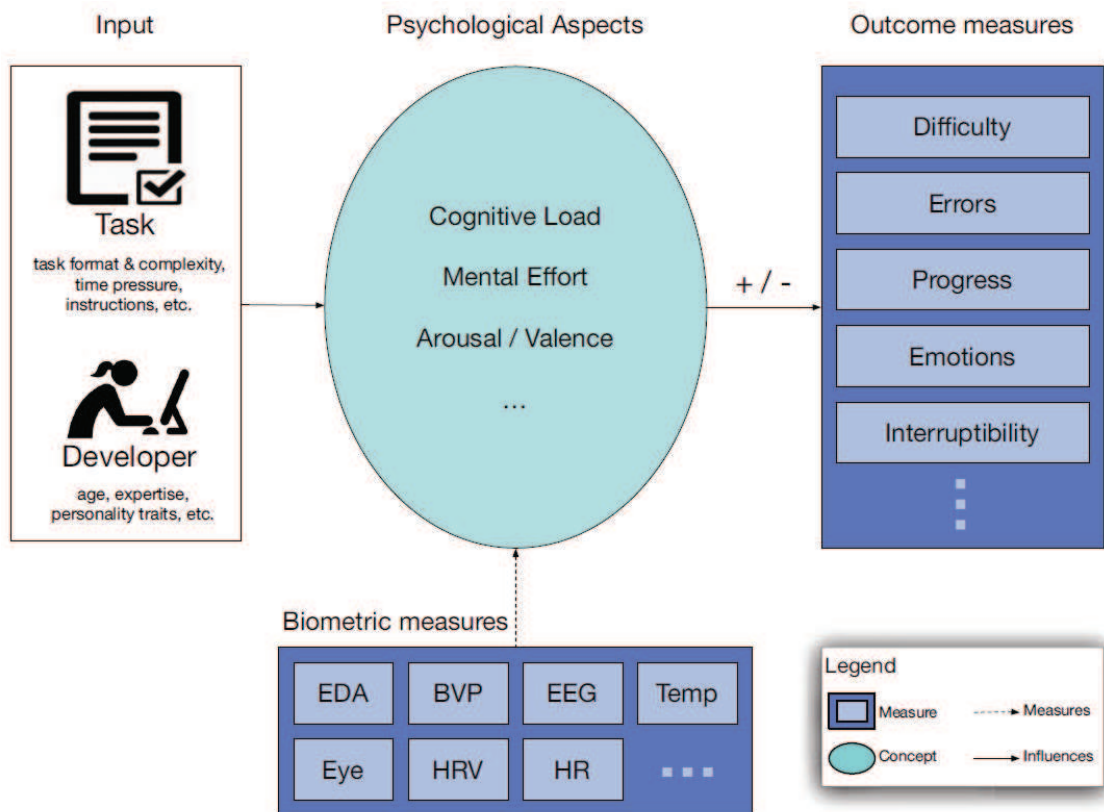


multiple mathematic methods to clean the noise and classify the outcome aspect for each behavior (PAAS et al., 2010; SWELLER; AYRES; KALYUGA, 2011). More advanced techniques that commonly are inaccessible, such as functional magnetic resonance imaging (fMRI), help comprehend how the activation pattern occurs in the brain from a software developer (AYRES, 2001; KO; MYERS, 2005; IQBAL; BAILEY, 2005; SIEGMUND et al., 2014a).

### 2.2.1 Biometric Data

Human characteristics have been of rising importance in computer science over the last few years. The area responsible for translating human data into metrics is known as biometrics (JAIN; HONG; PANKANTI, 2000; FRITZ; MÜLLER, 2016). Diverse studies have shown that the biometric data collected has a direct relationship with the psychological states and it could be extending in various contexts (MURUGAPPAN et al., 2007; MURUGAPPAN; NAGARAJAN; YAACOB, 2009). Figure 8 presents an overview of the psychophysiological process applied to the software development context and details their characteristics for each step to extract metrics from human factors.

Figure 8 – Psychophysiological process in a software development context.



Source: Fritz e Müller (2016)

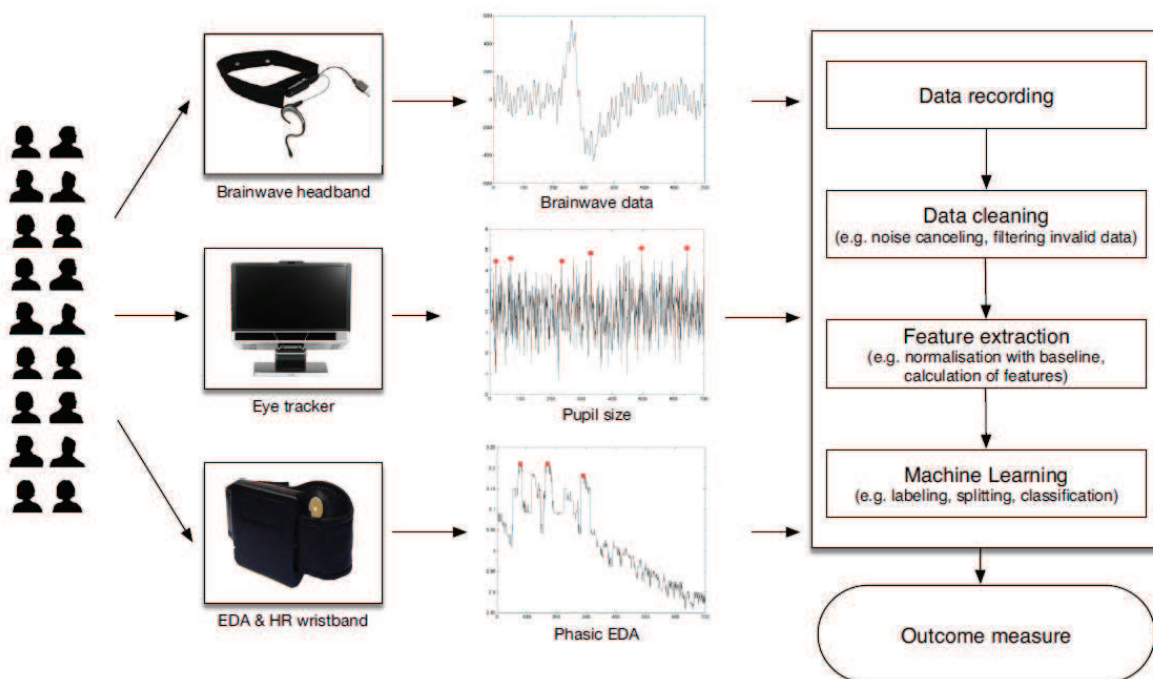
Biometrics is divided into five groups. First, breathing uses the respiratory rate (RR) to relate the respiratory rate with mental effort (KUZNETSOV et al., 2011). Second, the heart has three measurements; blood volume pulse (BVP), heart rate variability (HRV), and heart rate (HR) relating to different cognitive loads (RICHTER et al., 1998; PICARD; VYZAS; HEALEY, 2001). The third group measures eye features, such as movement, pupil size, and eye blinks to predict mental and memory load (SPRINGER, 2010; VENABLES; CHRISTIE, 1980).

The fourth group, the skin group, uses galvanic skin response (GSR) and skin temperature as a measurement to extract specific emotions (EKMAN; LEVENSON; FRIESEN, 1983). Lastly, the electroencephalogram measurement group is associated with physical and mental activity resulting in a workload rate and emotion classification (YOUNG et al., 2015; LI; LU, 2009).

### 2.2.2 Data Analysis

The long process begins once the biometric data is extracted. Collecting data is just the first step in several processes required to clean and classify the data in order for it to become a measurement. Figure 9 illustrates an overview of all the processes from beginning to end. First, raw data is recorded as it is generated for further analysis. After this, noise canceling and other techniques are applied to clean the data. Features are extracted and executed in machine learning or other techniques to generate the outcome measure of the collected data.

Figure 9 – Overview of the process to extract biometric data and produce measurements.



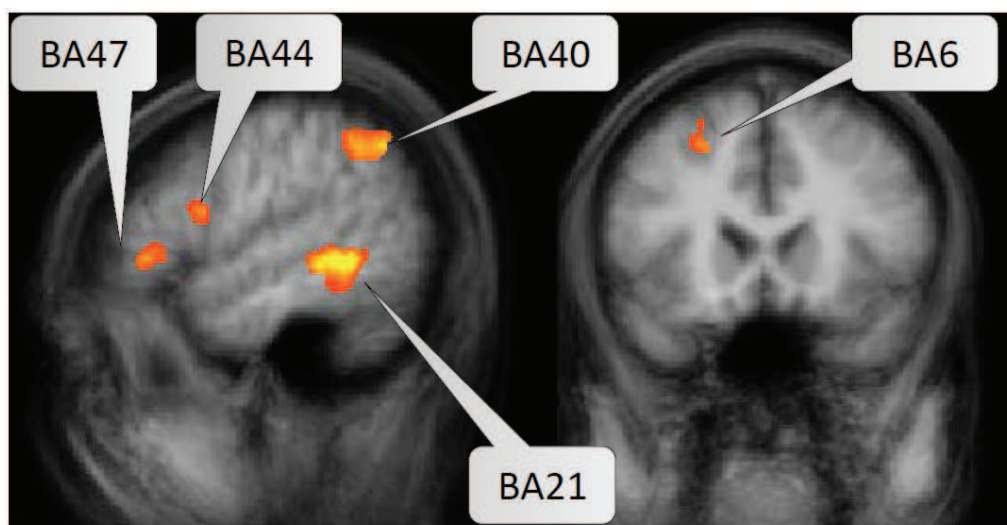
Nowadays, a large number of devices can collect data from our bodies with various levels of precision. The amount of data generated for simple scenarios can easily surpass millions of data points, and all this information needs to be handled for the next step (RAWASSIZADEH; PRICE; PETRE, 2014). Once recorded, the data is subjected to techniques such as noise cleaning and invalid data filtering to remove bad data inserted during the recording step (ZÜGER; FRITZ, 2015; MÜLLER; FRITZ, 2016).

Feature extraction is next, and it involves cleaning the data so it can be classified and normalized by individuals. In the EEG scenario, one option is to export a group of channels into five frequency bands (MURUGAPPAN et al., 2007). In general, several methods can be applied to classify and label the collected data. A widely used method is known as the machine learning and support vector machine (SVM) approach, and it helps to generate the outcome measurement (MÜLLER; FRITZ, 2015). Data analysis of psychophysiological indication requires a series of methods and processes to extract values from an enormous amount of data. The use of APIs or the machine learning technique becomes crucial in generating a precise measurement.

### 2.2.3 Developers Brain

In the cognitive neuroscience, methods such as fMRI can help us to observe exactly where the program comprehension happens using the human brain as a psychophysiological indication. Basically, the human brain is divided into 52 brodmann areas (BA) and each of these areas are associated with one or more characteristics (SIEGMUND et al., 2014a). Figure 10 shows five BAs that most become active while a developer is understanding a program which can also be used with electroencephalograph devices (JELLIFFE, 1910a; JELLIFFE, 1910b).

Figure 10 – Activation pattern from a developers brain while comprehending code.



Source: Adapted from Siegmund et al. (2014a)

Brodmann areas 6 and 40, as presented in the picture, are responsible for the cognitive process such as silent word reading, working memory for verbal and/or numerical material and also for division of attention being that all of them might take place directly when a developer is reading syntax and keeping values in their mind (BOTTINI et al., 1994; PRICE et al., 1994; VANDENBERGHE et al., 1997). On the other hand, Brodmann areas 21, 44, and 47 are related to word analysis and language processing which is responsible for semantic processing. This process then translates the meaning for each word and combines them in a multi-word level which generates chunks in the brain (SKOSNIK et al., 2002; BAHLMANN; SCHUBOTZ; FRIEDERICI, 2008; PETERSSON; FOLIA; HAGOORT, 2012).

Working memory plays a significant role in software development, and it is clearly represented on the BA capture. Keeping numbers and words in mind allows developers to execute the code in their minds superficially (BERG, 1948; SIEGMUND et al., 2014a). Consequently, program comprehension is directly connected to language processing, such as single words and symbols. These sets of instructions, once combined, will then generate an understanding of our brain using the chunks (FIEBACH et al., 2005; GRODZINSKY; SANTI, 2008). Simultaneously, some studies have shown that programmers are far better than average, usually magnificent, with their original language (DIJKSTRA, 1982).

Overall, studies have shown that there are patterns identifying how a developers' brain works on tasks that involve understanding a program (SIEGMUND et al., 2014a). However, it becomes tough to use an fMRI in an inexpensive and accessible way. To solve this problem, new methods, such as biometrics and, more specifically, electroencephalography can help bring this evaluation closer to the industry. At the same time, better ways of analysis are still required since non-intrusive devices will have more noise and external impacts.

### **2.3 Electroencephalography**

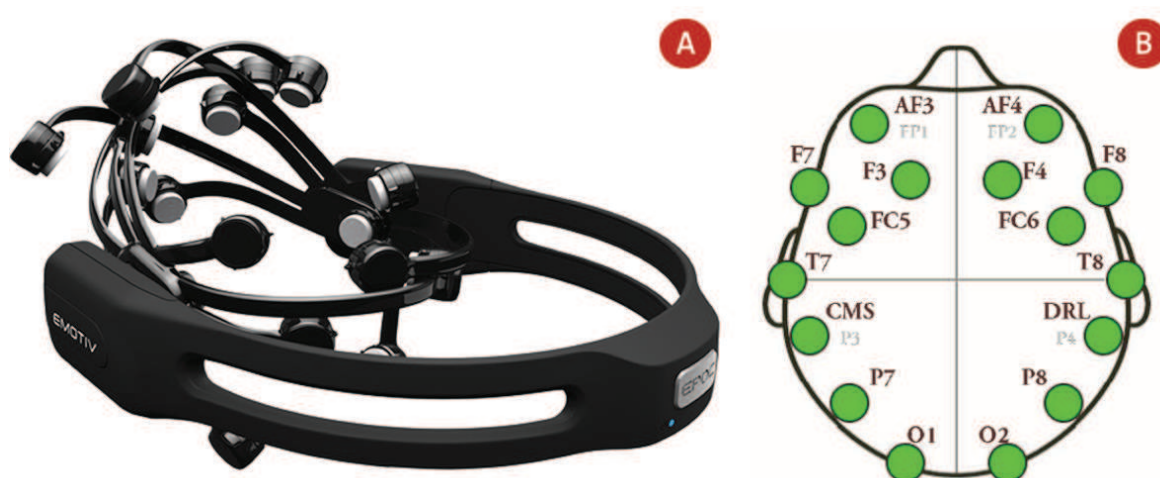
Electroencephalogram (EEG) has been used to record the discharge synchronized activity in the human's brain when large groups of neurons are active for nearly a century (BERGER, 1937). Discovered in 1924, the EEG has been used to evaluate rhythmic waves from various event-related potential (ERP) experiments that involve presenting a stimulus to a subject and after that collecting their brain activity (MCCARTHY; WOOD, 1985). The opportunity of eliminating obstacles and generating commands without them using muscles and nerves opens up a range of possibilities, raising the mental potential to the next level (NIJBOER et al., 2008).

EEG is considered a non-invasive method using electrodes; in other words, it does not require any surgical intervention like electrocorticography (ECoG). An electrode is composed of a small gold disk placed very close to the scalp and sometimes can be used with a liquid to amplify the sensitivity while collecting data. In essence, EEG allows recording a range of milliseconds resolution from brains' activity. The electrodes are put on top of the scalp and then it translates electrical signs generated by brain activity (5-100mV).



Once the brain wave is captured, it can then be divided into five distinct groups: delta (1-4Hz), theta (4-8Hz), alpha (8-13Hz), beta (13-30Hz), and gamma (30-70Hz). From these categories, the theta and alpha frequency bands are the most noteworthy waves for researchers (PACE-SCHOTT; HOBSON, 2002; KLIMESCH, 1999). Figure 11A presents an example of a non-intrusive EEG device using a wireless local area networking signal to transmit the information to a computer. A subset of 14 channels of the international 10-20 system of electrodes is presented in Figure 11B. Furthermore, EEG is prone to collect noise from muscle, oculomotor, and also ambient sounds, such as electrical devices (CRK; KLUTHE; STEFIK, 2015a).

Figure 11 – EEG device and the electrodes location by international 10-20 system.



Source: Emotiv Systems (2017)

As stated above, the delta wave is mostly present through deep sleep and contains a high level of noise present during awake experiments (PACE-SCHOTT; HOBSON, 2002). On the other hand, the theta wave is known to complement the alpha because the amplitude of one goes against the other, representing working memory performance (MATHEWSON et al., 2012). The alpha wave is known as the most used frequency during experiments, as it is a marker of cognitive inactivity and represents a lack of a task completion (KLIMESCH, 1996; RAY; COLE, 1985; JENSEN, 2002; COOPER et al., 2006; KLIMESCH; SAUSENG; HANSLMAYR, 2007; RIHS; MICHEL; THUT, 2007). Finally, the gamma wave is known for a huge amount of environmental noise and is not supported completely on all devices.

EEG devices generate 128 samples per second, making the alpha and theta peaks the most susceptible groups for cognitive effort (KLIMESCH, 1999). Conversely, the amount of mental effort being used in working memory is part of the cognitive load theory (CLT), which refers to the theta and alpha waves. Furthermore, the EEG signal is composed of diverse signal oscillations, which compound waveforms. This waveform may contain peak frequency, which is used to evaluate cognitive tasks and could be translated into a measure of program comprehension.

### 2.3.1 Waveforms

It is estimated that the number of nerve cells in the brain is approximately  $10^{11}$ , and since cortical neurons have a vast interconnection between other neurons, the surface of only a single neuron has between 1,000 and 100,000 synapses with other neurons (NUNEZ; SRINIVASAN, 2006). The synapses generate a voltage change that can fluctuate from -70mV over positive values when an action takes place in the brain. Meanwhile, a nerve impulse has an amplitude of 100mV, which lasts for about 1 ms (MALMIVUO; PLONSEY, 2012).

The behavior of the signal generated by an electroencephalogram device (EEG) can vary in multiple and different ways. A person's level of consciousness is closely related to the activity captured by the EEG and can have different meanings based on the context inserted. As presented in Figure 12, EEG waves can have different shapes and formats during their existence, which will intercalate a lower amplitude and a higher dominating frequency over time (Cooper, R., Osselton, J. W., & Shaw, 2002).

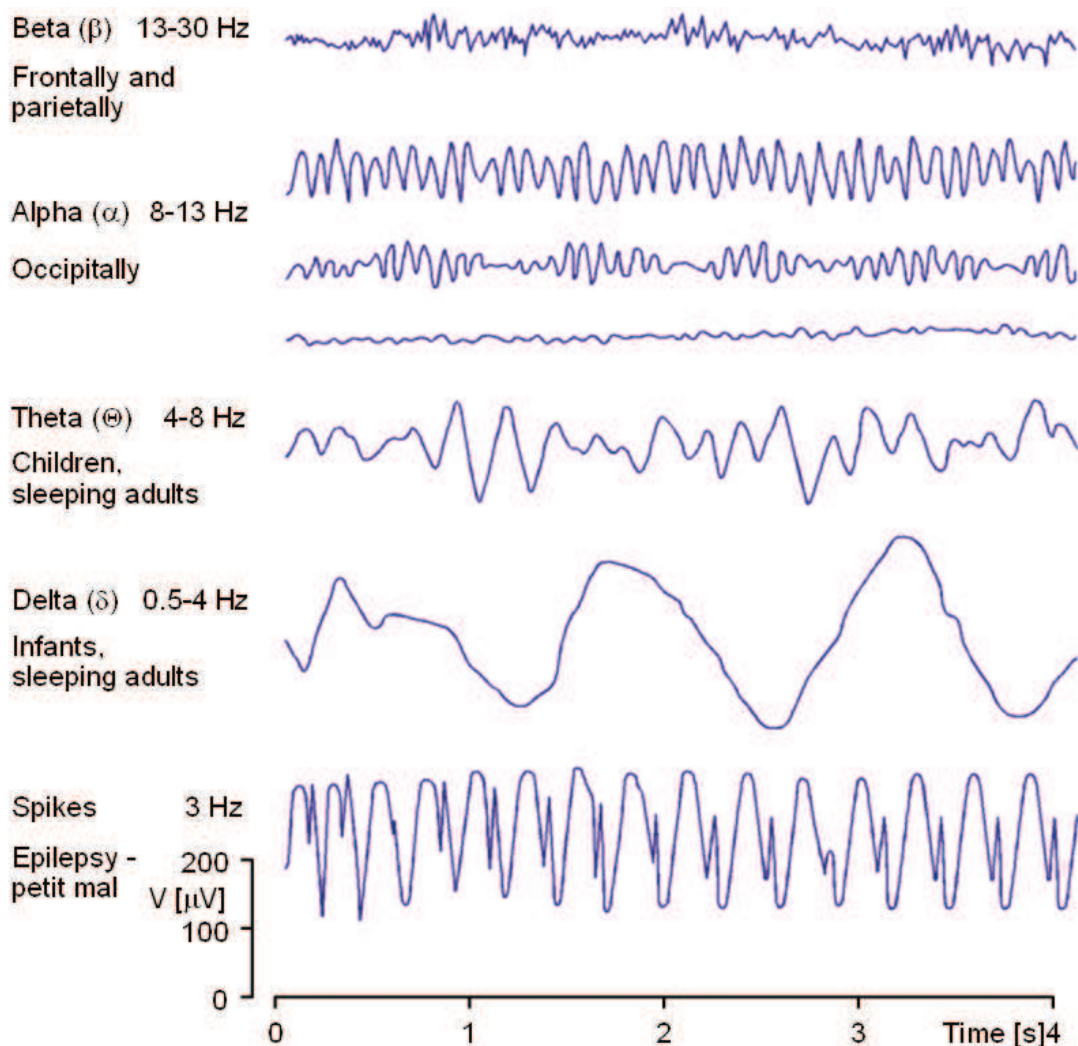
According to the literature, the alpha band power is known for increasing, or in other words, synchronizing, in the absence of a task. It is also known for decreasing, also known as desynchronizing, neurons when it attends a work request (KLIMESCH; SAUSENG; HANSLMAYR, 2007; RIHS; MICHEL; THUT, 2007). When developers keep their eyes open without executing a task, the alpha peak is known to be desynchronized and consequently decreasing the power peak (WELCH, 1967; KLIMESCH, 1999; CRK; KLUTHE; STEFIK, 2015a). It is known that the alpha wave is a key point to measure comprehension, and the next step sets the baseline of the peak alpha frequency, which is based on the individual as described by Equation 2.1 (KLIMESCH, 1996).

$$\text{Peak Alpha Frequency} = 11.95 - 0.053 \times \text{Age} \quad (2.1)$$

The individual peak alpha frequency (IAF) is calculated for each subject and the value generated is used as a baseline to calculate a subset defined in the IAF. Frequently, the subset groups selected have a range of 2Hz such as lower-1 alpha (L1A, covering from IAF-4Hz to IAF-2Hz) and lower-2 alpha (L2A, varying from IAF-2Hz to IAF) (PFURTSCHELLER; ARANIBAR, 1977; CRK; KLUTHE; STEFIK, 2015a). The intensity of an event can generate higher or lower cognitive work to be processed during and after the event. Moreover, the event makes a group of neurons oscillate in synchrony during a given task (KAUFMAN et al., 1990). Based on the synchronization and desynchronization event, the event-related desynchronization (ERD) can measure and reflect the cognitive load for a particular subject, which is presented in Equation 2.2 (KAUFMAN et al., 1990).

$$\text{ERD} = \frac{(\text{band power})_{\text{rest}} - (\text{band power})_{\text{task}}}{(\text{band power})_{\text{rest}}} \times 100 \quad (2.2)$$

Figure 12 – Sample of EEG waves grouped by type over time.



Source: Malmivuo e Plonsey (2012)

Equation 2.2 shows that more desynchronization is observed in the task, in other words, lower values in the band power in a period; the higher it is, the greater the attention demand for a given task. In this way, this type of analysis can support miscellaneous scenarios since it uses the baseline created by the individual himself, and, at the same time, handles varied environments where the software developer uses it (KLIMESCH, 1999). Recent studies using alpha ERD have shown an association among ERD and cognitive effort being a factor that indicates the higher level of alpha peak and alertness during task-specific (KLIMESCH; SAUSENG; GERLOFF, 2003). Another study indicates the use of ERD to estimate the domain-specific is more suitable than the mental ability throughout intelligence tasks that directly influence the subject's background (STIPACEK et al., 2003; GRABNER; STERN; NEUBAUER, 2003; KLIMESCH; SAUSENG; HANSLMAYR, 2007).

In essence, the working memory has a direct relationship between the level of ERD generated and its rest period. Being able to measure the group of neurons in the correct spot and process it based on the individual programmer might be the key to allowing a scale of how good or how bad a source code is using only the interpretation from a developer's understanding.

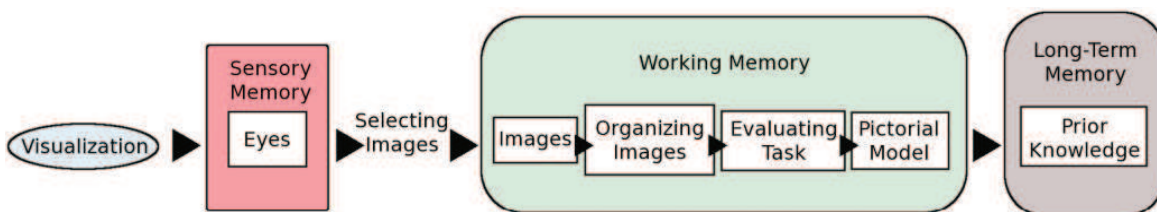
### 2.3.2 Cognitive Load

Cognitive load is known for having a close relationship with working memory and the aspect of short-term and long-term memory during decision making (BADDELEY, 1994; ENGLE, 2002). At the same time, program comprehension is present in many studies, playing an important role when validating different aspects in software development such as methodologies, adoption, and design paradigms (ARISHOLM et al., 2007; BORNAT; DEHNADI; SIMON, 2008; PENNINGTON, 1987b; MAYRHAUSER; VANS, 1994).

Working memory is defined for capturing, transforming, and executing information, which directly impacts comprehension activity (BADDELEY, 1994). Existing evidence has shown that alpha and beta waves have a connection with cognitive load while performing tasks (BAŞAR, 2012). The CLT model is based on a limited capacity of humans, and the closer our brain is to this limit, the more tired our brain becomes over time (SWELLER, 2014).

The long term is commonly known to support information in a limitless way since it can group the information in chunks and groups of chunks (BADDELEY, 2001; ANDERSON et al., 2011; SIEGMUND et al., 2014b; CRK; KLUTHE; STEFIK, 2015a). Figure 13 represents an experiment using an image as the input. The mental path illustrates which part is responsible to process the information as it passes through both the working memory and long term memory.

Figure 13 – An example of the cognitive process and working memory as a flow.



Source: Anderson et al. (2011)

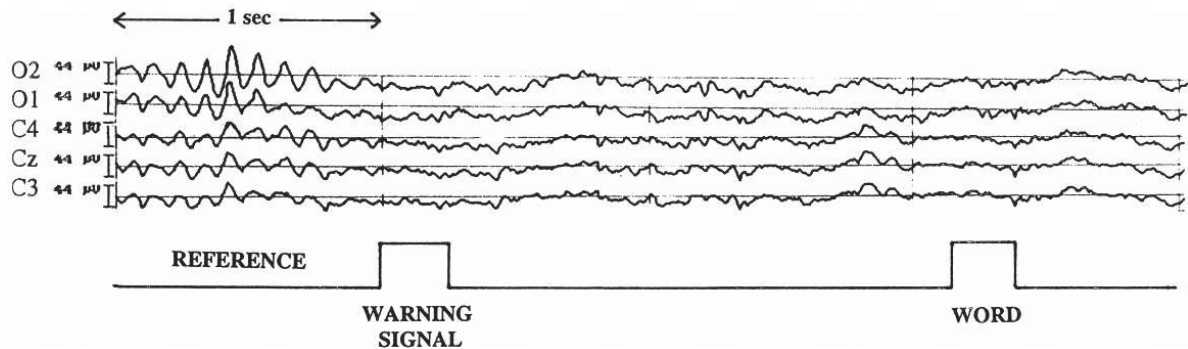
The theory of cognitive load can be broken down into three categories: intrinsic, extraneous, and germane (SHIFFRIN; SCHNEIDER, 1977). The intrinsic load focuses on the translation of the complexity of a task. At the same time, the extraneous is represented by the inappropriate designs of a task. Lastly, the germane load is related to the construction of cognitive schema based on repeatedly practiced tasks (GEVINS et al., 1997; CRK; KLUTHE; STEFIK, 2015a).

Recent studies have applied the concept of cognitive load in different fields, such as video



games, visualization effect, combat simulation, and virtual reality, amongst others (BERKA et al., 2005; GRIMES et al., 2008; COYNE et al., 2009; ANDERSON et al., 2011; MATHEWSON et al., 2012). ERD is a method that can be used to measure the cognitive level from a developer, using as a reference, a relax moment and as input words after each relax moment as illustrated in Figure 14.

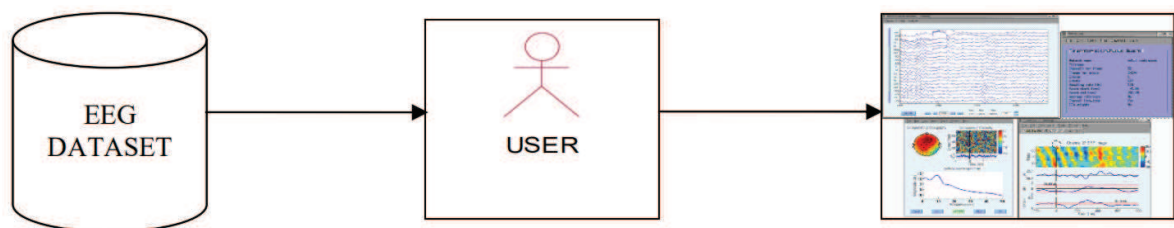
Figure 14 – Alpha desynchronization during cognitive load activity.



Source: Klimesch (1999)

The literature has presented an enormous amount of tools, such as EEGLab, which can be used to collect information from EEG, especially those that are related to the ERP concept. The information gathered is processed with a clear step, after extracting and finally generating statistics and values to estimate cognitive processes as shown in Figure 15 (DELORME; MAKEIG, 2004). The amplitude of EEG is measured in microvolts ( $\mu\text{V}$ ), and it normally uses a range of 10-50  $\mu\text{V}$  to detect brainwave pattern (GURUMURTHY; MAHIT; GHOSH, 2013). This pattern is often cleaned when loaded in EEGLab from devices such as EPOC+ from the EMOTIV brand.

Figure 15 – Summary of the brainwave process from a dataset.



Source: Gurumurthy, Mahit e Ghosh (2013)

The first step after obtaining the EEG dataset is applying the filtering process to smooth the frequencies and sharpen the brainwaves to make the sign clear and easy to interpret (GURUMURTHY; MAHIT; GHOSH, 2013). The first two filters that might be applied during the

process is high-pass and low-pass filtering. The first filter makes the wave sharper, and the last one smooths the brainwaves usually with frequencies at 0.1Hz and 60Hz frequencies respectively. After the application of the common methods, tools like EEGlab can support a wide range of techniques to clear data noise and extract features from the channels used in the EEG.

This background chapter provided the information used in this study. Starting with program comprehension with mental models, it moves to the psychophysical indicator which shows the relationship between mind and body. It finished with the electroencephalograph and the method for extracting a cognitive load from a human brain. The next chapter will discuss how this study searched for related works using a systematic methodology and what the future opportunities are in the field of program comprehension and software development.

### 3 RELATED WORK

This chapter focuses on investigating (1) a broad vision of state of the art in the (2) program comprehension area related to (3) cognitive effort. Even though the current literature has proposed many works over the last decade, little is known about the interpretation between program comprehension and cognitive effort. Also, knowledge and characteristics are missing in respect to whether cognitive indicators can be used to measure the level of program comprehension.

Most studies to date suffer from various limitations. The studies often lack a clear understanding concerning the different types of BCI techniques supported, fail to reveal the main influential factor related to cognitive effort, rarely describe the research strategies applied, and are limited to investigating specific issues as opposed to broader issues.

Additionally, it becomes crucial to study (1) which areas could be explored and has already been discovered, (2) how the gaps from the studies can be grouped together to get a bigger picture from this field, and (3) how the current methods can help us map potential areas for improvement. To address all these issues, this study proposes a systematic mapping study (SMS), whose methodological steps were designed and performed considering well-accepted guidelines such as (KITCHENHAM; BUDGEN; Pearl Brereton, 2011a; BUDGEN; BRERETON, 2006a; PETERSEN; VAKKALANKA; KUZNIARZ, 2015a).

Thereby, a systematic mapping study was developed following a carefully drawn protocol that underlies the experimental design for this mapping study. The rest of the chapter is organized as follows. Section 3.1 introduces the study methodology. Subsequently, Section 3.2 presents the results and interprets the findings achieved. Section 3.4 describes the set of research opportunities that can be explored. Finally, Section 3.5 elucidates the precaution taken into account while executing the SMS.

#### 3.1 Research Plan

This work explores gaps in the current literature and provides an understanding of cognitive comprehension applied to software abstraction in a broader context. In this sense, some works related to the subject have been identified (NICOLAS-ALONSO; GOMEZ-GIL, 2012; ZIGNEGO, 2014; ROZENBLIT, 2005; MÜLLER; FRITZ, 2015).

A review of brain-computer interface (BCI) devices and highlight their fundamental importance in current applications (NICOLAS-ALONSO; GOMEZ-GIL, 2012). They also discuss the advantages and disadvantages of BCI technology as well as presents an overview of the applications that BCI could explore. However, the study neither uses a systematic protocol nor analyzes the various types of BCI devices that academia has adopted in cognitive applications and empirical studies.

Similarly, Zignego (2014) analyzes the commands used to control wheelchairs and also summarizes the human factors utilized for the selection of a protocol for BCI commands. They

focus on the cognitive signals and influential factors applied to wheelchair devices instead of program comprehension. Furthermore, Rozenblit (2005) summarizes the concepts of cognitive computing, the impact it has on the design of intelligent systems, and their complex requirements. However, this study does not aim at mapping the software abstractions and BCI devices that other studies may have explored.

The systematic mapping studies produced by literature are not centered around the cognitive effort in program comprehension (ZHI et al., 2015). Rather, they investigate the effort estimation of software development (GUEVARA; FERNÁNDEZ-DIEGO; LOKAN, 2016) and software documentation (ZHI et al., 2015). To summarize, none of the produced works focus on adopting a systematic mapping study protocol to investigate the cognitive effort in program comprehension.

This study is one of the first to report the gaps in the current literature, point out trends of research, and classify publications and research strategies that are related to the cognitive effort in software comprehension. For this, the following sections seek to make a mapping study protocol to find and classify the candidate studies, regarding the BCI interfaces produced, the independent variables investigated, the research strategies utilized, the type of software abstraction, and the influential factors that affect cognitive effort.

### 3.1.1 Research Strategy

The objective of this work is twofold: (1) to provide a classification and a thematic analysis of the current literature considering the interplay between program comprehension and cognitive effort; and (2) to identify research directions for future investigations. For this, six research questions (RQ) are defined to investigate different facets of these two objectives. These RQs are described as follows. Table 1 summarizes these six RQs, their motivations, and the variable investigated, as well as provides an overview on how they address the gaps previously mentioned.

***RQ1: What BCI devices are most often used in studies concerning the comprehension of programs and UML diagrams?*** As previously mentioned, several BCI devices have been used over the last few years. Controlled experiments in many research fields use different kinds of BCI devices for capturing the brain signals, e.g., Emotiv EPOC, NeuroSky, Traditional Electroencephalogram, among others. This study looks to understand what kind of BCI devices that are often preferred (or most used) by both academia and industry. This issue also helps identify the motivation that allow the researchers to choose a specific BCI device for their experiments. Overcoming this issue will give us a clear indicator of the type of BCI devices that both the academia and industry have judged to be more important to support.

***RQ2: What is the state of the art in empirical studies concerning the comprehension of programs and UML diagrams?*** A set of widely accepted research methods can be used to evaluate the current literature, including, for example, controlled experiments, case studies,

Table 1 – Research questions for systematic mapping study.

Research Question	Motivation	Variable
RQ1: What BCI devices are most often used in studies concerning the comprehension of programs and UML diagrams?	Find out the types of BCI devices utilized and explored that have been widely adopted.	BCI device supported
RQ2: What are the research strategies used in the current empirical studies concerning the comprehension of programs and UML diagrams?	Discover the research method studies on the comprehension of programs and UML diagrams that have been used to investigate the problem.	Research strategy
RQ3: What are the variables investigated in the empirical studies?	Understand and comprehend which independent variables were usually explored to measure the comprehension.	Variables investigated
RQ4: What software abstractions are used in the current work?	To know and analyze the software artifacts, and abstractions studies were focused on.	Software abstraction
RQ5: What factors can influence the comprehension of software abstractions?	Reveal the main reasons responsible for impacting on the comprehension of software artifacts.	Influential factor
RQ6: Where have the studies been published?	Scrutinize and analyze the relation of the main source of publications, publications type, and publications per year.	Research venue

Source: Created by the author

quasi-experiments, and surveys, as well as other widely-recognized methodologies. However, little is known about how these methods have been applied in this research field. For this reason, we focus on scrutinizing how academia has conducted investigations into the cognitive efforts involved in software comprehension. This paper will provide data to evaluate the maturity of this research field. The literature may have data about the effects of software abstraction on mental states. The key concern behind the investigation of mental states in software engineering is understanding the factors that impact the developers' productivity given that many factors are responsible for disrupting their comprehension, making them more error-prone and less efficient (FARIAS et al., 2015a). As a matter of fact, studies that are not classified under a research method have not been adequately evaluated. This implies that they may not be reproducible or be useful in real-world / practical software projects.

**RQ3: What are variables investigated in the empirical studies?** There is no specific variable to measure developers. Instead, in experimental studies, researchers and investigators make use of a set of indicators and indexes. The indicators are applied to show the developers' perception of the tasks. However, the use of various variable settings can change the context among experimental studies. While the combination of the number of correct answers with the engagement variables indicates the comprehension in using the new technique, the time effort and frustration variables may indicate the impact of environment pressure during a software activity. For this, these questions seek to define sets of variables researchers utilized to interpret the understanding in these contexts. It becomes clear that a wide view on the variable used is needed

to know what challenges are still left unexplored.

**RQ4: *What software abstractions are used in the current work?*** Working on a software project, developers deal with many types of software abstractions, such as design diagrams and source code. Little is known about the methods used to achieve comprehension. This question investigates which software abstractions have been used. We look at reporting the purpose and the context in which such abstractions were used.

**RQ5: *What factors can influence the comprehension of software abstractions?*** The interest in analyzing the cognitive comprehension is focused on understanding the factors and circumstances that may compromise a developer's cognitive understanding in dealing with specific abstractions. This RQ is crucial because it may indicate the proper task according to the specific level of experience. The experience level is a factor that can affect the cognitive perception. The degree of detail and structural complexity are only 2 of many variables that affect the developers' comprehension (BORSTLER; PAECH, 2016). This question seeks to identify and classify the factors used in the literature. This RQ will analyze the influences as well as side effects that the issue produces in the source code.

**RQ6: *Where have the studies been published?*** The candidate studies are usually published in different sources, for example, conference proceedings, journals, book chapters, and workshops. Also, each of these sources has a different reputation level. To know and analyze these aspects of candidate studies is a key concern to define the maturity of the research field. It enables to point out the approximate year that this area of the investigation emerged, the year that publications reach a higher frequency, and future tendencies regarding the production of articles about the cognitive effort in program comprehension. This question focuses on where the studies have been published, and understand how these publications spread over the years, and through the publication modalities.

After describing the research questions, the next step is to draw the strategy defined to retrieve potentially relevant articles available currently. For this, well-known guidelines (advocated in Budgen e Brereton (2006a) and Petersen, Vakkalanka e Kuzniarz (2015a)) regarding the construction of search strings (SS) and the definition of search scope were followed. Thus, it was possible to specify an unbiased and iterative search strategy to systematize the selection of candidate articles, which were narrowly related to our research questions. To indicate the existing literature reviews, including empirical studies, systematic literature reviews, mapping studies, surveys, as well as evaluate them quantitatively and qualitatively, the search strategy was elaborated and refined, thereby, allowing mitigate threats to the construction of our search.

### 3.1.2 Data Extraction

Given that the quality of the search strategy depends on how well the SS were defined, we adopted the PIO (population, intervention, and outcome) criteria. The PIO has been effectively used and reported on in numerous previous empirical studies (e.g., (BUDGEN; BRERETON,



2006a; QIU et al., 2014)). Using the PIO criteria the RQs can be decomposed into individual facets so that a set of synonyms, abbreviations, and alternative spellings can be formulated.

Having these facts on hand, the SS could be built making use of Boolean operators such as “AND” and “OR”. The first term, *population*, refers to the technologies, devices, and standards related to this study, e.g., BCI and EEG. The second, *intervention*, considers specific issues in the context of software methodology or techniques, e.g., source code, design models, programming language, software modeling, and UML (Unified Modeling Language). The third, *outcome*, deals with main factors for practitioners, e.g., improved modularization, enhanced comprehension, and superior understanding.

More specifically, they refer to the level of cognitive load required to comprehend software abstractions, or even the degree of modularization to leverage a superior understanding of design decisions. As the topic being investigated has rarely been explored, there is no concern about restricting the search terms which were taken into consideration.

The search terms were defined based on the following steps. First, the major key terms are defined. Secondly, alternative words are identified, such as synonyms or related terms. Next, keywords are checked whether the ones chosen are in fact found in a representative sample of widely known articles related to the topics being investigated. Associate synonyms and alternative words narrowly related to the main keywords using the Boolean operator “OR” and, finally, relate the major terms with the Boolean operator “AND”.

Table 2 – Search strings used to retrieve the candidate articles.

<b>Major terms</b>	<b>Alternative terms</b>
BCI	EEG, Electroencephalogram, Brain-computer interface
Comprehension	Understanding, Interpretation, Perception, Comprehensibility, Understandability, Misinterpretation
Software	Source code, Design models, Diagrams, UML Programming language, Modeling
Empirical	Experiment, Survey, Case study, Action research

Source: Created by the author

Although many combinations of the search terms may be formulated, the candidate articles retrieved are often similar. After performing several searches and analyzing the results, the most significant results were obtained using the SS presented as follows:

( *Electroencephalogram OR EEG OR “Brain-computer interface” OR BCI* ) AND  
 ( *Understanding OR Comprehension* ) AND  
 ( “*Programming Language*” OR “*Source Code*” OR *Diagram OR UML* ) AND  
 ( “*Empirical study*” OR “*Case study*” OR “*Controlled Experiment*” )

Table 3 shows eight electronic databases (study research scope) used to retrieve the potentially relevant articles. By doing so, the SS defined was adapted and applied to each database listed as listed in Appendix C. These databases were chosen because they have been widely used in several review studies previously published (QIU et al., 2014; FERNÁNDEZ-SÁEZ; GENERO; CHAUDRON, 2013)). Additionally, the most relevant journals, conferences, and workshops within the area of computer science. Note that only studies published in such electronic digital libraries were taken into consideration. Additionally, the Scopus database was not used because it is a payed database and at this moment only free academic source were used.

Table 3 – Databases used to find candidate studies.

<b>Electronic Databases</b>	<b>Website Link</b>
ACM Digital Library	<a href="http://dl.acm.org/">http://dl.acm.org/</a>
CiteSeer Library	<a href="http://citeseerx.ist.psu.edu/">http://citeseerx.ist.psu.edu/</a>
Google Scholar	<a href="https://scholar.google.com.br/">https://scholar.google.com.br/</a>
IEEE Explore	<a href="http://ieeexplore.ieee.org/">http://ieeexplore.ieee.org/</a>
Inspec	<a href="http://digital-library.theiet.org/">http://digital-library.theiet.org/</a>
Science Direct	<a href="http://www.sciencedirect.com/">http://www.sciencedirect.com/</a>
Springer Link	<a href="http://link.springer.com/">http://link.springer.com/</a>
Wiley Online Library	<a href="http://onlinelibrary.wiley.com/">http://onlinelibrary.wiley.com/</a>

Source: Created by the author

### 3.1.3 Inclusion and Exclusion Criteria

The Inclusion Criteria (IC) and Exclusion Criteria (EC) were established to filter the articles found in the initial search. These criteria were defined based on the guidelines described in (PETERSEN; VAKKALANKA; KUZNIARZ, 2015a; QIU et al., 2014; BUDGEN; BRERETON, 2006a). Table 4 and Table 5 present the inclusion and exclusion criteria used respectively. Only studies written in English and published up to December 2016 were considered.

Extracting data from the potentially relevant articles is crucial as one of the purposes of the systematic mapping study is to produce evidence in a domain to be plotted with a high level of



Table 4 – The inclusion criteria used to filter the studies retrieved.

<b>Inclusion Criteria (IC)</b>
IC1: articles written in English.
IC2: articles available in scientific journals, conferences and workshops.
IC3: articles published through December 2016.
IC4: articles that matches with the search strings.

Source: Created by the author

granularity (PETERSEN; VAKKALANKA; KUZNIARZ, 2015a; KITCHENHAM; BUDGEN; Pearl Brereton, 2011a; BUDGEN; BRERETON, 2006a). This subsection defines the strategy to accurately collect the information from the selected studies. For this, numerical values, nominal or ordinal data will be produced, which are pivotal to any meta-analysis regarding the objective and research questions.

Table 5 – The exclusion criteria used to filter the studies retrieved.

<b>Exclusion Criteria (EC)</b>
EC1: the title, abstract or even any other part of their content were closely related to the search keywords, however without any semantic interplay, were removed.
EC2: articles that are patent registration, or are not published in English (the default language considered in our study), or are in initial stage, typically presenting an abstract and summary of future steps.
EC3: articles with title having no term defined in the search string, or even the meaning of the title is completely contrary to the purpose of the issues addressed in the research questions.
EC4: studies with abstract that do not address any aspect of the research questions.
EC5: studies that are duplicate.
EC6: studies which do not address issues related to the use of neuroscience methods to grasp how developers comprehend software abstractions.

Source: Created by the author

Figure 16 shows the form elaborated to synthesis the data collected (FERNÁNDEZ-SÁEZ; GENERO; CHAUDRON, 2013; ŠMITE et al., 2010).

Figure 16 – Form used for data extraction.

Article's Data							
Title							
First Author							
Source Name							
Year of publication							
Type of publication		Conference		Journal		Workshop	Others
Research Question's Data							
BCI Device							
Dependent Variables							
Research Strategies		Experiment		Survey		Technical Briefing	Others
Software Abstraction							
Influential Factors							
Distribution of Studies							

Source: Created by the author

Data extraction can be considered an error-prone task as the authors might extract different data from the studies, considering their mentally held indicators. To mitigate this risk, this study has elaborated a data extraction form, which serves as a guide to unambiguously collect and combine the data gathered. The keywords used to recover information from the selected studies so that the RQs can be properly answered are detailed in Table 6.

Table 6 – Keywords used to recover information from the selected studies.

RQ	Variable	Terms
RQ1	BCI device supported	ActiveTwo (AT), actiCap/g.USBamp (AU), Electro-Cap/g.USBamp (EC), and Emotiv Epoc (EE)
RQ2	Research strategy	The methods found in the candidate articles are the following: Experiments, surveys, case study and technical briefings
RQ3	Variables investigated	Attention, Comprehension, Correctness, Difficulty, Efficiency, Experience and Workload
RQ4	Software abstraction	Arithmetic tasks, Business Process Models, Class Diagrams, Component Diagram, Motor abilities, Sequence Diagrams, Test Reading, and Source code
RQ5	Influential factor	BCI Limits, Complexity, Distraction, Multiple, and Subject Experience
RQ6	Research venue	conferences, workshops, and journals

Source: Created by the author

Quality Assessment (QA) has been conducted in the selected studies. The QA guarantees not affecting key requirements for replication in other studies, i.e., do not influence the interpretation of the published work. For this, we defined some issues based on previous mapping

studies Qiu et al. (2014), Fernández-Sáez, Genero e Chaudron (2013) and literature review Kitchenham, Budgen e Brereton (2010a). Appendix B presents the results obtained. Nine Quality Assessment (QA) questions are specified and described as follows:

- QA1:** Is there a clear statement of the goal of the research?
- QA2:** Is the context description well-defined?
- QA3:** Does the work carefully describe the related works?
- QA4:** Have the authors produced a clear explanation about cognitive load?
- QA5:** Did researchers discuss the threats to validity?
- QA6:** Did the authors write well-defined conclusions?
- QA7:** Have the authors evaluated the results?
- QA8:** Have conclusions complied with the objectives and/or research questions?
- QA9:** Did the authors suggest further research directions?

### 3.2 Study Filtering

After presenting the study methodology, this subsection aims to outline the study filtering process used to select candidate articles so that a set of relevant primary studies can be done. The quality of the work is appraised whether it can be, in fact, used to answer research questions investigated (Appendix B). In total, the filtering process has nine steps, which were defined to select the primary studies carefully. Below each step is described following the order of the filtering process, which are explained as follows:

- **Step 1: Initial Search.** Collect the search results after submitting the search string to the digital libraries. In total, 1709 studies were identified.
- **Step 2: Remove Impurities.** Remove discrepancies obtained from the search results. For this, the exclusion criteria EC1 and EC2 are combined. Regularly, a call for papers of conferences, special issues of journals, patents specifications, and research reports were examples of improperly retrieved papers.
- **Step 3: Filter by Title.** Filter studies by applying the exclusion criteria, EC3. Thus, we discarded any study whose title had no term found in the search string.
- **Step 4: Filter by Abstract.** Filter studies considering EC4. Therefore, we discarded any of the studies whose content is not related to the key issues addressed.
- **Step 5: Combination.** All the filtered studies from the last step were combined together.
- **Step 6: Duplicate Removal.** Usually, a study can be found in two or more digital libraries. Thus, we applied EC5 for removing all duplicates, thereby assuring the uniqueness of each study.

- **Step 7: Study Addition by Heuristic.** Although the search mechanisms of the digital libraries were widely recognized, occasionally some works could not be found. Thus, we added such studies manually to our sample of the primary studies.
- **Step 8: Filter by Full Text.** We filtered some studies by applying EC6 to the full text, excluding studies that were not relevant to the subject model comparison or matching.
- **Step 9: Representative Work Selection.** Define the final list of the primary studies.

Following is a series of data refined after completing the systematic review, totaling 12 primary studies, which were selected through a very strict research process. Figure 17 illustrates the results obtained in each step for the filter step described above. The complete list of primary studies is presented in Appendix A, which includes a unique identifier, prefix S, followed by a numeric value, from 1 up to 12, e.g., S01, S02, etc.

Figure 17 – Phases for the filtering process in the systematic mapping review.

	Initial Search	Impurity Removal	Filter by Title	Filter by Abstract	Combination	Duplicate Removal	Addition by Heuristics	Filter by Full Text	Representative Work Selection
ACM Digital Library	122 <i>5.73% filtered</i>	115 <i>76.52% filtered</i>	27 <i>74.07% filtered</i>	7	64	62	66	13	12 <i>0.70% Remain</i>
CiteSeerX Library	459 <i>37.69% filtered</i>	286 <i>92.30% filtered</i>	22 <i>77.27% filtered</i>	5					
Google Scholar	631 <i>45.64% filtered</i>	343 <i>83.96% filtered</i>	55 <i>61.81% filtered</i>	21					
IEEE Explore	13 <i>0.00% filtered</i>	13 <i>93.28% filtered</i>	9 <i>30.77% filtered</i>	3					
Inspec	20 <i>0.00% filtered</i>	20 <i>50.00% filtered</i>	10 <i>80.00% filtered</i>	2					
Science Direct	308 <i>26.30% filtered</i>	227 <i>71.81% filtered</i>	64 <i>79.69% filtered</i>	13					
Springer Link	136 <i>14.71% filtered</i>	116 <i>50.86% filtered</i>	57 <i>77.19% filtered</i>	13					
Wiley Online Library	20 <i>100.00% filtered</i>	0 <i>0.00% filtered</i>	0 <i>0.00% filtered</i>	0					
<b>Total</b>	1709 <i>34.45% filtered</i>	1120 <i>78.21% filtered</i>	244 <i>73.77% filtered</i>	64 <i>3.12% filtered</i>					

Source: Created by the author

A total of 1709 candidate studies were collected before applying any filtering criteria as (1). The next step was to remove impurities by applying EC1 and EC2, which discarded 34.45% (589) units from the sample (2). Then, the EC3 was implemented to the remaining studies (1120), 78.21% being filtered out through (3) title review. Next, the EC4 was applied to the remaining 244 studies, and 73.77% were discarded after reading the abstract (4). Therefore, a relevant sample of 64 works was obtained.

Next, (6) six duplicate studies were removed by applying the EC5, 3.12% (2/64). Then, four works were added by applying by (7) applying heuristics, producing a sample of 66 studies, or increasing it by 6.45%. Then, the (8) EC3 and EC4 were applied to the 66 studies, resulting in 80.3% (54/66) being discarded. After examining the remaining papers, 7.7% (1/13) were excluded because it did not meet the (9) quality criteria for relevance criteria. Finally, 12 studies were selected as the most representative, hereafter called *primary studies*. Their main findings are discussed after their critical analysis.

### 3.3 SMS Results

This section describes the results after analyzing the data collected. Each question below has a detailed explanation and which papers use the most common categories in each RQ.

**RQ1: BCI device** This RQ looks at the type of BCI devices that are often preferred by the academia and industry as previously mentioned. This issue also helps identify the BCI devices which have been used over the last years. Controlled experiments in many research fields often use different kinds of BCI devices for capturing the brain signals, e.g., EPOC Emotiv, NeuroSky, Traditional Electroencephalogram, among others. Table 7 presents the mapping between the BCI devices and the primary studies selected.

Primarily, the results indicate that studies are using BCI devices that have up to 14 channels to capture the data from the users brain waves. For this, the main finding is that the most Brain-computer interfaces (50%, 6/12) used in research studies (reviews, and experiments) are the product line of EPOC Emotiv and Neurosky. Therefore, the findings of these experimental studies are derived from the analysis on 14 brain waves channels.

Table 7 – RQ1: BCI device supported.

BCI Supported	Quantity	Percentage	List of papers
Emotiv EPOC	4	34%	[S08], [S09], [S11], [S12]
Neurosky	2	17%	[S03], [S04]
ActiveTwo	1	8%	[S02]
actiCap/g.USBamp	1	8%	[S01]
Electro-Cap/g.USBamp	1	8%	[S05]
None	3	25%	[S06], [S07], [S10]
Total	12		

Source: Created by the author

It also indicates a relationship between the cost and the adoption of the Brain-Computer-Interfaces in the experiments. While the EPOC Emotiv price is up to \$799 USD currently

(Emotiv Systems, 2017), Neurosky remains in an experimental version and then it will be available to final customers (Neurosky, 2016). This simple project has a restricted budget. However, as this is an emerging research field, the academia are still evaluating the results.

Further, an important number of studies used advanced BCI, i.e., a total of 3 studies (24%) used BCI with more than 32 channels, reaching until 280 channels. These BCI devices are ActiveTwo (8%, 1/12), actiCap/g.USBamp (8%, 1/12), and Electro-Cap/g.USBamp (8%, 1/12) specifically. This means some studies are seeking to conduct a deep and precise analysis.

**RQ2: Research strategy** The aims to scrutinize how the investigations regarding cognitive effort applied in program comprehension have been conducted in academia. A set of widely accepted research methods is used for this purpose generally, i.e., controlled experiments, case studies, quasi-experiment, and surveys. Moreover, this question can disclose the maturity of this research field, through the analysis of what have extent the kind of research methods have been used. Table 8 presents the results regarding which aspects were used to compare the models.

Table 8 – RQ2: Research strategies per empirical method.

Empirical method	Quantity	Percentage	List of papers
Experiment	8	67%	[S01], [S02], [S03], [S05], [S08], [S09], [S11], [S12]
Case Study	0	0%	-
Survey	3	25%	[S06], [S07], [S10]
Technical Briefing	1	8%	[S04]
Total	12		

Source: Created by the author

Results show the experimental bias of studies concerning cognitive effort on program comprehension. The majority of works (67%, 8/12) are controlled experiments. It was observed that none of the works were validated in real-world settings. Instead, they were focused on academic solutions. These solutions affect the closeness of mapping cognitive dimension.

Another interesting finding is that there are only three surveys (25%, 3/12) produced in this research field. These results suggest that there is little research direction, and the experiments produced are not based on their guidelines. Furthermore, there is an absence of case studies (0%, 0/12), thus failing again in providing a validation close to the domain problem.

**RQ3: variables investigated** This aims to investigate the independent variables that researchers have adopted to specifically measure the developers cognitive effort. This is due to many aspects regarding the program comprehension being evaluated. For this, we expect the following independent variables initially: Attention, Correctness, Comprehension, Difficulty, Efficiency, Experience and Workload. Table 9 presents the types of independent variables primary studies used.

Table 9 – RQ3: variables investigated.

Independent Variables	Quantity	Percentage	List of papers
Attention	1	8%	[S09]
Comprehension	0	0%	-
Correctness	2	17%	[S01], [S02]
Difficulty	2	17%	[S03], [S04]
Efficiency	1	8%	[S10]
Experience	2	17%	[S11], [S12]
Usability	1	8%	[S08]
Workload	1	8%	[S05]
None	2	17%	[S06], [S07]
Total	12		

Source: Created by the author

The most investigated independent variables are related to mental effort directly. The majority of studies (50%, 6/12) used the Correctness (16.6%, 2/12), Experience (16.6%, 2/12) and, Difficulty (16.6%, 2/12) independent variables specifically. This kind of variable is utilized to verify how difficult an artifact is to mentally process. On the other hand, the remaining independent variables were related to productivity. A considerable number of studies (33.3%, 4/12) investigated program comprehension on the perspective of the Efficiency (8.3%, 1/12), Usability (8.3%, 1/12), Attention (8.3%, 1/12), and Workload (8.3%, 1/12) independent variables. These type of variables are usually used to verify how a technique or language affects the user's productivity.

**RQ4: Software abstraction** This looks for an answer to the type of software abstraction, i.e., tools, program and modeling languages, documentations, among others that those empirical studies which the developer's comprehension on software abstractions have been related with. Table 10 details the abstractions mapped according to these primary studies.

First, the experiments focused on evaluating cognitive effort on software development. The majority of studies (41%, 5/12) focused on investigating the cognitive effort of the source code. The source code is the main artifact developer's deal with during the software production cycle. This study could not find any other studies attempting to solve direct mental operations regarding the source code, i.e., How much mental processing lies at the notational level, instead of at the semantic level?. Additionally, they still lack on investigating error-proneness in the artifacts, i.e., what extent does the language or model increase the chance of the user making errors?

Second, a considerable number (41%, 5/12) of primary studies focused on the evaluation of cognitive efforts of manual activities over the artifacts. The arithmetic notation (17%, 2/12),



Table 10 – RQ4: software abstraction.

Software abstraction	Quantity	Percentage	List of papers
Arithmetic task	2	17%	[S01], [S02]
Motor	1	8%	[S10]
Source Code	5	41%	[S03], [S04], [S08], [S11], [S12]
Test reading	2	17%	[S05], [S09]
None	2	17%	[S06], [S07]
Total	12		

Source: Created by the author

the motor notation (8%, 1/12), and test reading notation (17%, 2/12) artifacts were evaluated specifically. However, we did not find any evidence by researchers investigating the modularity level effects and how the parts of the notation are identified, accessed by the brain. Finally, only two works did not mention any software abstraction since they are surveys and reviews, i.e., they are not evaluating any criteria presented in this question.

**RQ5: Influential factors** This seeks to map and classify the factors that academia have been studying. For example, the experience level is a factor that can affect the cognitive comprehension. The level of detail, structural complexity, and amongst others, are variables that also effect the developers comprehension. Table 11 shows the influential factors mapped according to the primary studies.

Table 11 – RQ5: influential factors.

Influential factors	Quantity	Percentage	List of papers
BCI Limits	1	8%	[S06]
Complexity	3	25%	[S01], [S02], [S05]
Distraction	3	25%	[S03], [S04], [S08]
Multiple	2	17%	[S07], [S10]
Subject Experience	3	25%	[S09], [S11], [S12]
Total	12		

Source: Created by the author

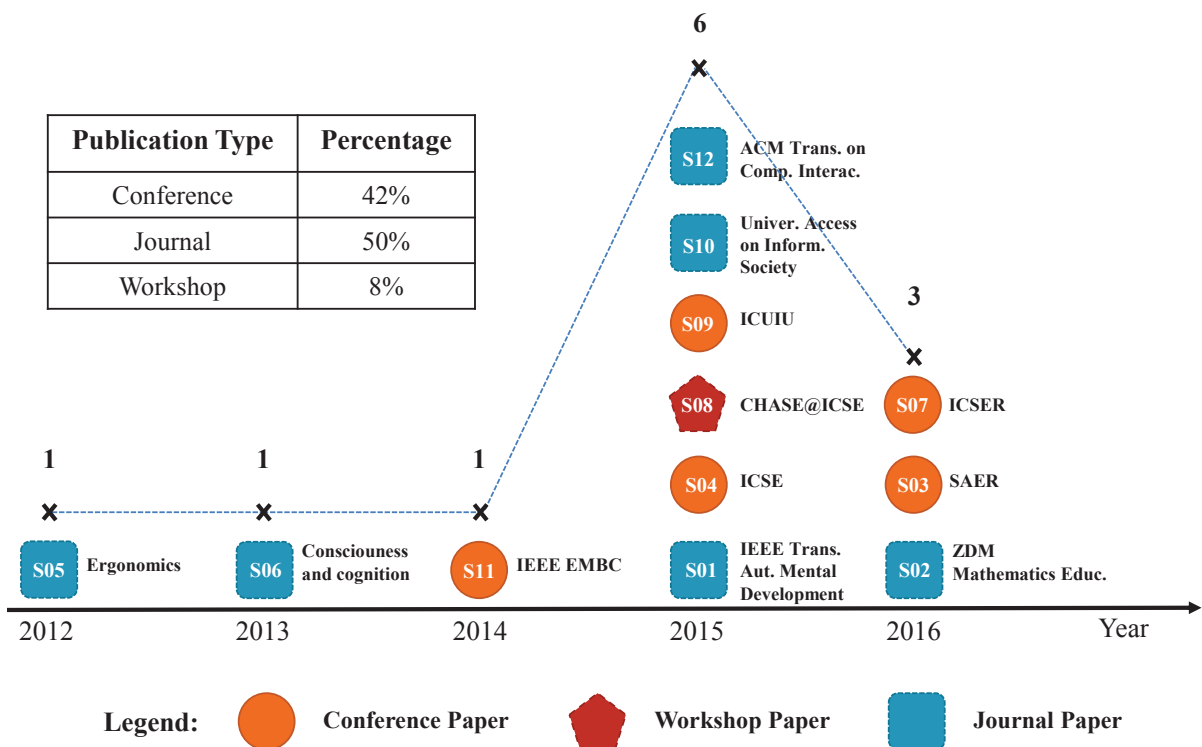


The majority of works (50%, 6/12) focuses on subject aspects such as the subject experience (25%, 3/12), and distraction (25%, 3/12). This indicates the user’s abilities was extensively implied in the effects in comprehending software abstraction. Furthermore, the aspects regarding the user surroundings were investigated significantly. Thus, the BCI Limits (8%, 1/12), and Complexity (25%, 3/12) are the influential factors that were also investigated. Finally, a few studies (17%, 2/12) focused on multiple influential factors, rather than expanding the scope.

**RQ6: Research venue** This attempts to determine where the candidate studies were published, i.e., to know the main conferences, journals, book chapters, and workshops names. It enables to point out the year that publications reach the higher frequency and future tendencies about the cognitive effort in program comprehension.

Figure 18 shows the distribution of primary studies published per year from 2012 to 2016. The dashed line represents the number of studies published in each of the years. No primary studies were found before 2012 and were rarely explored until 2014. On the other hand, the number of publications increased significantly beginning in 2015. Since 2014 some BCI devices that have an interface ease to manipulate, and collect data from sensors have been popularized.

Figure 18 – RQ6: Primary studies distribution by year.



Source: Created by the author

The percentage of primary studies according to the source of publication is also highlighted in Figure 18. A significant part of primary studies, i.e., 50% (6/12) were published in journals. One paper (S05) published at Ergonomics in 2012, and one (S06) at Consciousness and

cognition in 2013. Next, three primary studies were published in journals in 2015, i.e., One study (S01) published at IEEE Transactions on Autonomous Mental Development, one (S10) at Universal Access on Information Society, and another one (S12) at ACM Transactions on Computer Interaction.

The last one (S02) was published in 2016 in ZDM Mathematics Education. Furthermore, an applicable percentage (42%, 5/12) of studies were issued in conferences. In 2014, one study (S11) was released at IEEE EMBC. Next, two studies were issued in 2015, i.e. one (S04) were issued at ICSE, and another one (S09) at ICUIU. Two other primary studies were published at conferences in 2016, i.e., one (S03) published at SAER, and the last one (S07) at ICSE. Finally, only 8% (1/12) of primary studies were presented at Workshops, i.e., this one (S08) at CHASE workshop held at ICSE conference in 2015.

To summarize, a significant amount of publications in this research field have been created in the last few years. The studies that were published have been published from 2012. Finally, all these factors combined implies that these studies provide a reliable basis for further research.

### 3.3.1 Comparative Studies

This systematic mapping study (SMS) overcame the gap regarding the lack of a panoramic view about the studies performed that use BCI to measure detectable indicators of brain activity while developers carry out regular software engineering activities such as program comprehension or software abstraction.

A robust protocol was defined to guide the whole experimental process by specifying search strings, describing inclusion and exclusion selection criteria, combining automatic and manual searches in eight recognized electronic databases, and assessing the selected studies qualitatively. An accurate filtering process retrieved a sample of 1,709 candidate articles, which were carefully filtered, classifying 12 of these as primary studies.

Four new conditions were applied to the 12 articles selected as primary studies in order to select articles relevant for this study. First, (1) only papers that worked with BCI devices were used and (2) studies which had at least some independent variable defined in their study. Also, (3) studies that have some level of relationship with software abstraction. Besides, (4) case studies were not selected as the purpose of this research is to isolate the comprehension process; (5) articles which have multiple influence factors were not considered as the goal of this paper is to focus on scenarios where only one factor is present.

After exploring the primary studies, six research questions were using to draw up some findings and also to pinpoint fields where there is a shortage of publications. Also, five of the six research questions were used as criteria to examine and summarize the primary studies. Additionally, a brief description of each filter applied is described below, followed by the options available for each.

**BCI device support** Nowadays there are a few BCI devices in demand, the ones that were found relating with software comprehension are: (1) Emotiv EPOC with 14 electrodes and running at 256Hz; (2) Neurosky with only one electrode and sampling at 512Hz; (3) Biosemi ActiveTwo EEG Systems with 32 or 64 wired electrodes and up to 16-kHz sampling; (4) acti-Cap/g.USBamp using a cap and with 16 slots for electrodes; (5) Electro-Cap/g.USBamp from 8 to up 64 channels and also using a cap to connect each electrode.

**Research strategy** Present the study strategy employed in the paper. The option found in the SMS are: (1) experiment, where a controlled experiment was executed to isolate a variable; (2) survey, where the academic literature was thoroughly reviewed; (3) technical briefing, giving insights on technical information regarding this area.

**Variables investigated** Describe which variables were manipulated in order to affect the dependent variable: (1) attention is a cognitive process of selectively concentrating whereas disregarding other information; (2) comprehension is the capability to absorb and interpret; (3) correctness is the conformity from what is the truth or what is expected, which is error free; (4) difficulty is a variation in tasks with the same goal, but complex paths; (5) expertise, how background knowledge could reduce the probability of error; (6) usability, how easy or how difficult it is to perform a task; (7) workload, how much mental effort in the brain is used to do a specific task.

**Software abstraction** These are methods that can represent more meaning with less visual information, such as (1) arithmetic task, which executes mathematical tasks and provides an answer in a strict way; (2) source code, understanding and executing the software code in the brain; (3) test reading, analyzing if the user paid attention after reading it.

**Research venue** Different levels of variations may influence the final result. The factors used are (1) complexity, where a task has various levels of difficulty to complete; (2) Distraction, not allowing a person to concentrate while performing a task; (3) subject experience, validating if a developer's experience can provide assistance for better results for the same level task.

After defining a group of criteria to evaluate each primary study, Table 12 represents each category using four different icons. In the legend, the symbol (●) means that the study supported that evaluation, the semi-circle (◐) means that option is partially supported. The empty circle (○) represents that it is unsupported and the cross-box (⊠) means it is not applicable. Table 12 displays a comparative analysis of the approach and methods used to compare the nine papers selected. Three studies have been removed based on the five new rules defined above.

This work is the first to use solid, systematic mapping rules in the main search engines in the area of computer science to portray the state of the art of program comprehension and BCI. It has been identified that these studies do not accomplish problems which involve different levels of comprehension and point to problems to measure program comprehension in the software engineering area.

Table 12 – Comparison of related works.

Primary Studies	BCI Device					Strategy			Variable						Software			Venue			
	Emotiv EPOC	Neurosky	ActiveTwo	actiCap/g.USBamp	ElectroCap/g.USBamp	Experiment	Survey	Technical Briefing	Attention	Comprehension	Correctness	Difficulty	Expertise	Usability	Workload	Arithmetic Task	Source Code	Test Reading	Complexity	Distraction	Subject Experience
S01	○	○	○	●	○	●	○	○	○	○	●	○	○	○	○	●	○	○	●	○	○
S02	○	○	●	○	○	●	○	○	○	○	●	○	○	○	○	●	○	○	●	○	○
S03	○	●	○	○	○	●	○	○	○	●	○	●	○	○	○	○	●	○	○	●	○
S04	⊗	⊗	⊗	⊗	⊗	○	○	●	○	○	○	●	○	○	○	○	●	○	○	●	○
S05	○	○	○	○	●	●	○	○	○	○	○	○	○	○	●	○	○	●	○	○	○
S08	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	●	○	○	○	○
S09	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
S11	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
S12	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

**Legend:** (●) Supported (◐) Partially Supported (○) Not Supported (⊗) Not Applicable

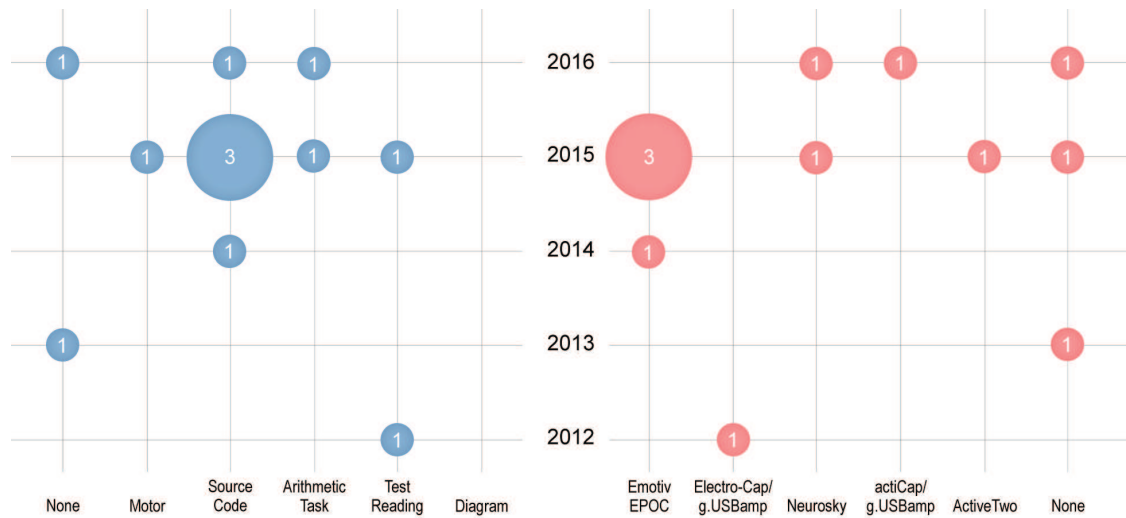
Source: Created by the author

### 3.4 Research Opportunities

This section further discusses challenges, utilizing as support a bubble graph with the primary studies distributed over the years. Figure 19 summarizes the percentage of primary studies in intervals from January 2012 through December 2016. The graph shows the number of papers published increased considerably, 83.33% (10/12) from 2014 onward. These studies mainly used a device from the Emotiv brand, presumably because it was simple to apply in the real environment. The source code was the most studied artifact, but it also displayed that different areas only appeared in one study. The number shown in Figure 19 highlights that it is a new area in computer science, which displays a potential to grow focus on how we process and what we interpret from software as a user or developer.

The main results from this systematic mapping study are as follow: (1) Most studies use low-cost but effective Brainwear Wireless EEG Technology to provide a mobile brainwave-sensing headset. (2) Also, the studies focus on understanding the effects of the cognitive processes on the correctness and difficulty of running development tasks. (3) Besides, the primary studies report empirical results rather than propose new cognitive indicators. (4) The studies analyze how test scripts and source code are processed in the developers' brains. However, none of the

Figure 19 – Combining RQ2, RQ3, and Year.



Source: Created by the author

selected studies have investigated how much the hard cognitive process of software abstractions lies at the notational level as opposed to the semantic one or even if it monitors the cognitive load. Finally, in order to discover the emotional responses, more research is required to identify when a developer achieves his/her peak mental performance.

Furthermore, software quality could be measured based on the difficulty of readability and comprehension that developers have by reading source code (BORSTLER; PAECH, 2016). As new methods emerge from neuroscience and are applied in computer science, such as fMRI, this might suggest that new ways to measure program understanding might help to evaluate more precisely how a language is interpreted for a developer (SIEGMUND et al., 2014a).

### 3.4.1 Cognitive Measurement

To truly understand the current literature, this subsection gives research directions about cognitive measurement based on ideas from the primary studies collected by the systematic study. Many other articles show how this area can become the future of how we evaluate and understand what a developer is thinking and how this affects the quality and productivity from a developers' perspective. The evolution of the current software languages and what we know about language interpretation can be greatly improved upon by using biometrics, especially from the brain, to enhance the future of program languages. Each area that may improve the cognitive measurement is established as follows:

**Code and cognitive load** Several areas have been studied in relation to how language is processed in our brain, and this raises many questions: Do we all understand source code in the same way as we understand human language? Since software maintenance is done by humans,

and not by a machine, how does each programmer understand the same source code based on their level of expertise? (SIEGMUND, 2016). Metrics for the cognitive load could help a development team understand if a source code, system diagrams, or software architecture have various abstraction levels or modularization, based on the knowledge of each person in the group, and also which rules have the lowest overhead in the software industry.

**Developers efficiency** Another opportunity assesses to support developers and their work through less interoperability and more productivity. Preventing bugs and reducing lost attention are some of the benefits when the cognitive state is not interrupted from its focus mode. Notifying the team that a developer should not be interrupted has the potential to make the team more productive. Individual developer support, on the other hand, might improve the full capability of a developer to use in the best way his or her abilities, achieve better performance, and optimize highly demanded tasks, reducing costs and giving real-time recommendations when the developer is stuck (MÜLLER; FRITZ, 2015; FRITZ; MÜLLER, 2016).

**Perception in software** The design principles for notations, known as cognitive dimensions, plays a new role in software development. It has the potential to dramatically change how we interpret visual information from a developers perspective (GREEN; PETRE, 1996). For instance, how different levels of abstraction can help how a developer understands source code or diagrams, how syntax language is processed in our brain, and which combination is the most suitable to be applied to each environment? (CRK; KLUTHE; STEFIK, 2015a).

**fMRI in developer's brain** The BCI technology is helping to measure the electrical activity in our brain and apply its output in computer science. Experiments using Top-Down comprehension could help us to understand how a developer uses its family domain knowledge when developing. Also, it could verify which areas in the brain are more actively contrasted with the bottom-up comprehension methodology. Areas like design partners might be studied to see if they alter our brain activity and if some design patterns generate more or less stress in our brain activity. Another interesting point is how our brains work when we implement software compared with when we understand the source code (SIEGMUND et al., 2014a).

**Individual brain necessities** In general, the current literature has proposed techniques in the computer science education field for better ways to teach programming language. Even though we can test people using regular exams, such as multiple choice, we are not sure how the learning process happens within the brain, especially for developers, and also, how to identify which teaching technique is most suitable for each individual developer. With what we know regarding software quality, there remains many questions on how we comprehend complex tasks versus simple ones or how overloading our working memory affects the developer over long periods (CRK; KLUTHE; STEFIK, 2015b; SIEGMUND, 2016).

**Ethical and privacy concerns** Another interesting area is related to privacy and ethical concerns in supporting data collected from BCI devices, especially EEG's. Because this type of device generates huge amounts of information, the current hardware, in most of the cases, is only able to send and receive information without any particular security. This type of technol-



ogy could be used in the employee recruiting process as a way of measuring the expertise of a specific skillset or task. Another scenario where the developer might be evaluated and measured against other developers or global baselines. (SIEGMUND et al., 2014a; SIEGMUND, 2016; CRK; KLUTHE; STEFIK, 2015b; EVERS; SIGMAN, 2013; FRITZ; MÜLLER, 2016). Humans scarcely control their psychophysiological reactions to a specific scenario, which removes human choice; this area should be explored and understood as to what the limit should be for BCI technology inside business companies.

### 3.4.2 Measuring Comprehension

In the same way, program comprehension might be measured through EEG when a programmer reads the source code. Also, developers' comprehension could also be measured in software diagrams, taking advantage of methods such as the UML. Further, the results from this measurement could be used inside a subversion (SVN) system, where the developing effort to understand a source code or a diagram would be saved trace code as a metric, and it could be used to suggest where the system could be enhanced. Further, each area that may spend the measuring comprehension are defined as follows:

**EEG applied in tools** Another path of possible research would be to get insights on how different programming languages behave in the same developer in the comprehension and development stages. In general, the syntax of each language is similar, but the fact that having, for instance, static or dynamic type systems could change the way we find the solution for a specific problem (CRK; KLUTHE; STEFIK, 2015b). Also, EEG could be used to detect how programmers understand an artifact and, as an output, could influence how an IDEs behaves or highlight a row if a candidate for refactory or if the business logic is not clear. Colors can improve, but it is just a small piece of the overall cognitive process, which involves comprehension before taking any action, especially in a production environment (SIEGMUND, 2016).

**Neuroscience in program comprehension** Neuroscience opens up an unexplored area focused on trying to understand what and how we think during the software development cycle. Metrics could help a software company to choose the best technology based on each developer's effort and then scale to measure the developers' software comprehension effort. This study intends to develop a technique to identify the developers' effort to understand source code by using an electroencephalogram. This method should measure the developer's brain intensity when understanding the source code and deliver a metric that can help quality systems.

Finally, this study can pave the way for a more ambitious agenda regarding the challenges in better comprehending (1) why and how indicators of cognitive processes can help to improve current software development tools (2) to what extent the software abstraction notations may influence cognition, increasing (or reducing) the chance of developers' making a mistake. The (3) electrical activity in the developers' brain can indicate the understanding of the semantic (or role) of each architectural component and the application.

### 3.5 Limitations of the Evaluation

The SMS executed may be impacted by threats to validity that range from internal and construct validity, the validity of statistical conclusions to external validity (BIFFI; TUISSI, 2017). Only external validity was not considered in the mapping because it cannot be generalized for the industry. For this reason, this subsection aims at discussing how to minimize construct, internal, and conclusion threats and offers suggestions for the improvement of future work.

**Construct validity** Systematic mapping studies are well-known for not guaranteeing the inclusion of all the relevant works in the field (ORIOLE; MARCO; FRANCH, 2014; FERNÁNDEZ-SÁEZ; GENERO; CHAUDRON, 2013; QIU et al., 2014). This point can cause a possible lack of the keywords defined in this work and other relevant papers. Therefore, relevant works can also be present in a search engine that we may not consider. To bypass major problems related to these issues, rigorous procedures for retrieving and filtering primary studies were adopted. Keywords and their synonyms were defined according to well-established methods in (BUDGEN; BRERETON, 2006b; PETERSEN; VAKKALANKA; KUZNIARZ, 2015b). This study also included the main scientific databases (ACM Digital Library, CiteSeer Library, Google Scholar, IEEE Xplore, Inspec, Science Direct, Springer Link, and Wiley Online Library). Finally, to ensure the appropriate removal of duplicate studies, articles from all search engines were gathered in a separate directory before filtering the repetitive ones.

**Internal validity** This refers to the validity of the analysis performed if such conclusions derived from the data are internally valid (JULNES, 2004). Each primary study identified the factors, as well as the influences in cognitive comprehension, applied in software development. To categorize them, the hierarchical structure was proposed in (ALTMANNINGER; SEIDL; WIMMER, 2009; MENS, 2002). Based on these categories, it looked at each technique and then classified them, i.e., mapping each technique based on matching. Understanding and then mapping each technique can be seen by itself a threat to validity. With this in mind, any concerns throughout the study was to make sure that the selected primary studies were identified and analyzed in a consistent manner. Based on the exploratory nature of our study, the internal validity of the results cannot assure comparability to the more precise manipulation of independent variables found in controlled experiments, in which several influential factors are critical, e.g., the randomization of the subjects, safeguards against confounding factors, and so on. However, this extensive control cannot be applied to our study.

The conclusion validity threats are related to problems that can affect the reliability of our findings. For this, the study rigorously followed the steps provided by well-disseminated systematic mapping study protocol (PETERSEN; VAKKALANKA; KUZNIARZ, 2015b; KITCHENHAM; BUDGEN; Pearl Brereton, 2011b; KITCHENHAM; BUDGEN; BRERETON, 2010b; BUDGEN; BRERETON, 2006b). The study describes all the steps for conducting the systematic mapping study. Finally, all the conclusions in this article were made after collecting the results, avoiding the validity fishing and the error rate problem (BIFFI; TUISSI, 2017).



## 4 ACTIVITY RELATED NEURAL INDICATORS (ARNI)

The objective of this chapter is to present a model to estimate comprehension from source codes using signals collected by EEG devices. The activity-related neural indicators (ARNI) model is a technique that uses several methods and processes to measure program comprehension. Among the components used, there are four main parts of the proposed model. The (1) alpha peak method is used per developer, and the (2) power spectral density (PSD) is used to calculate the density of the wave based on the (3) fast fourier transform (FFT). The results are the input of the (4) ERD algorithm as the output is a metric for each source code snippet.

The outcome generated by the model is considered the measure of program understanding in a software development language, which is customizable by snippet code, developer, and environment level of precision. To explore the research opportunities listed in Section 3.4 and meet the navigation objectives in Section 1.4, this chapter proposes a technique to identify and classify brainwaves from a developer while they are reading and understanding source code.

This chapter is organized into six sections, as follows: Section 4.1 presents the necessary requirements to be able to identify and estimate waveforms in a development context. Section 4.2 describes various concepts used to detect and process the EEG signal. In the Section 4.3, it is present pseudo-code used in the models to preprocess and calculate the outcome from the model. Section 4.4 shows the *CognitiveEffort* technique used in the ARNI model. Section 4.5 describes how the architecture was developed and how the technique was created using different technologies. Finally, Section 4.6 describes several aspects of the model proposed.

### 4.1 Requirements

This section describes each element related to the functionality and purpose needed in ARNI. Chapter 3 highlighted several points that other studies had partially addressed and must be further studied. The items described below, as a primary focus, support and quantify the developer's perception of source code while providing an understanding in the software developer context. Further, each step requirement was broken down and defined as follows:

**Source code and EEG** Being able to identify the code displayed on the screen in most cases can be considered a tricky task. This functionality centers its goal on the capacity to detect the segment of source code from its totality in a software tool. This requirement is essential since the displayed code on the screen has a direct bearing on the cognitive area of a developer while the latter is developing software. Since the data collected from an EEG device comes, in most cases, from 128Hz up to 2048Hz entries per second, it becomes essential for the precision to be able to detect context switches in code at a range of milliseconds. As operational systems (OS) have minor differences in their clocks, even in milliseconds, this functionality must use a non-dependent and embedded system to trace these context changes. Also, the OS should be validating periodically to validate the synchronized among the tools used.

**EEG data capture** A traditional EEG device typically consists of a cap, electrodes, several wires, and an amplifier. Every part has a distinct purpose and behavior, the implication is that the developer is in the same position while wearing a cap with electrodes. From the developers perspective, it becomes impossible to move around while wearing such a device. A wireless EEG device becomes necessary when working in environments with agile methodologies such as Scrum, Kanban or Lean. Furthermore, the EEG device used must integrate all traditional components into one to be able to measure a developer in the real environment.

**Pre-processing and noise detection** The brainwave is known as non-stationary data, where the statistical characteristics of signals are not similar over time. In other words, as regards to the wave characteristics in the EEG data collected, changes are based on the cognitive behavior, and this impacts how we should be processing the data. The statistical term might include information such as mean, variance, frequency, and covariance, amongst others. Since EEG data has a lot of overlapping frequencies, techniques such as FFT should be used to preprocess the information in a non-stationary manner. While the FFT process is known as a method for transforming information without loss, other techniques are required so as to detect and remove noise inserted during the software development tasks. Another common interference is known as the blink of an eye and muscular movements while development is being done. These also need to be handled during programming.

**Identify mental behavior** This functionality primarily describes the requirements for the model to adapt itself to work for any software developer using any source code. Since every software developer has their own way of thinking and understanding the source code, it becomes crucial to be able to estimate and identify the brainwaves for individual developers. In this case, the model developed must classify different wave formats and pattern amplitudes related to a software developer's brain.

**Measuring program comprehension** This specification can be considered the main requirement in this study and its focus is on the translation of waveforms into a metric to measure program comprehension. Based on cognitive behavior, this metric should be able to translate the electric pulse from a group of neurons when the cognitive action occurs in the brain. Further, this process should use the cognitive level of relaxation and task execution to measure different intensities of understanding generated by the brain.

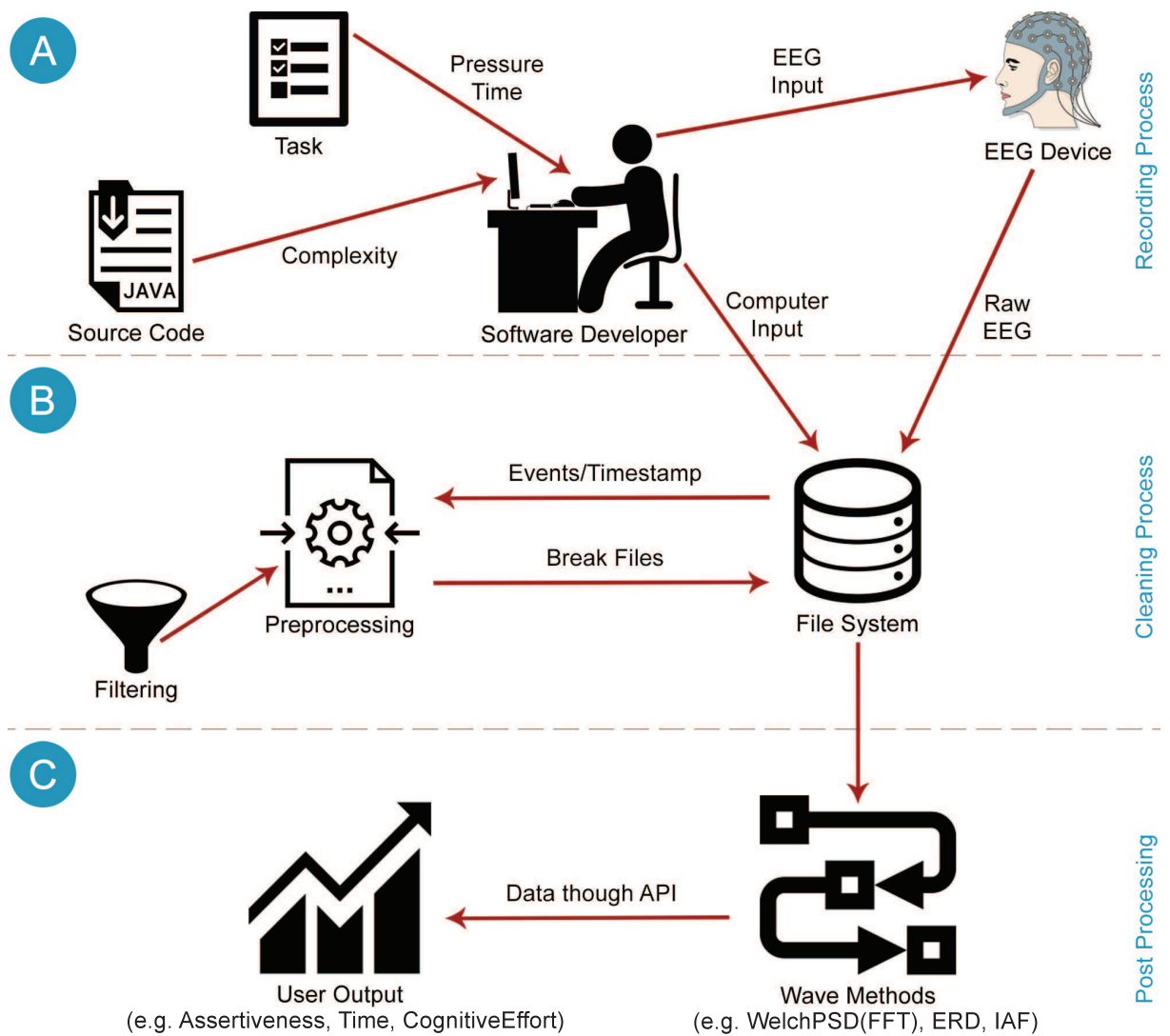
**Multi-platform and open-source** In the last few years, several OS were created and became relevant in the software developer environment. Interdependency from the OS brings a broad range of benefits and enables advantages such as maintenance and smooth process. The model process should be open-source so as anyone has the ability to contribute. This might bring gains such as higher security levels, quality enhancements, customization, among others.

This section focused on describing the requirements for the proposed model. Each element brings separate characteristics, which make up part of the EEG-Based model in this study. In section 4.2, each requirement listed above will be complemented with a description of concepts to better define and enhance their purpose within the model.

## 4.2 Concepts

This section will describe the concepts involved for each requirement raised previously in Section 4.1. Various conceptions were chosen to improve the comprehension of the solution adopted to accomplish the ARNI model goal. Figure 20 shows a wide-angle view of the model proposed in this study. The model process is divided into three main steps: the recording process, the cleaning process, and post-processing. The stages are well-defined and divided into several tasks, each with their own related activities and processes. Further, in order to understand better the details and characteristics inside of the ARNI model, each step in the process is broken down and defined as follows:

Figure 20 – Overview of the model processes.



Source: Created by the author

**Source code and EEG** This concept stands for the main task known as markers, shown in Figure 20A. A marker is a label created when an action occurs in the developer's system to allow event interconnection in uncommunicable software systems.

EEG generates at least 128 entries per second, and the precision has to be recorded at the millisecond. However, the marker must be inserted directly into the EEG data as this is the only way to ensure it is positioned in the appropriate event trigger by computer inputs. At the same time, another back-end process will keep tracing the logs and relate it to the actual event, in which case, source code is displayed on the screen.

**EEG data capture** As presented in Figure 20A, the concept of data capture concerns the fact that an EEG device requires the ability to work without wires and any other device except the headset itself. Nowadays, technologies such as Wi-Fi and Bluetooth are alternatives to transmit a considerable amount of data in small and middle-sized workspaces. This study uses the model EPOC+ from the brand EMOTIV to collect and store 256Hz of raw data, including the added markers in a file system inside the OS.

**Pre-processing and noise detection** This concept relates to techniques and processes that are used after the raw EEG data capture as presented in Figure 20B. The EEG data generated by EPOC+ is converted from the European data format (EDF) to comma-separated values (CSV). The first process takes into account two types of filters. The low-pass filter is responsible for filtering out high frequencies and the high-pass filter for filtering out low frequencies. Additionally, noise is identified through the mean value in each channel and then extracted from itself. This process removes the DC offset, which makes some types of wave patterns visible and human-detectable. Lastly, the data processed is saved again and the OS cleaned. The saved files are divided by the markers that were inserted in the EEG data.

**Identify mental behavior** After noise removal in the brainwave data, two methods are applied to convert the EEG data based using the wave pattern as presented in Figure 20C. The first step is to create windows with 125 milliseconds of size in the EEG data. After this, each window is used as an input to the FFT method to quantify the information based on wave frequency, which generates a PSD value. Next, the PSD value is used as a parameter to an ERD equation that uses the mean of all rest periods against each task period. To put it simply, this process calculates the neuron activity related to the current task.

**Measuring program comprehension** Figure 20C presents the last step of the process. The ERD is the input data used in this phase. According to the way each developer uses the model, it adapts itself to support different developers and software languages. Also, an understanding scale is created based on the assertiveness level, time, and the ERD information focused on measuring program comprehension. Each comprehension level created in the model has its own characteristics. Hence, the probability of understanding source code is attributed to each level of the assigned developers' mental states.

**Multi-platform and open-source** The developed technique formulated in this study executes on top of a MATLAB platform, and it is a multi-platform OS executable such as Linux,

MacOS, and Windows. Additionally, the source code created will become available in the web-based Git software known as GitHub, in other words, a version control system (VCS) in the Internet. The source code developed is published under the Massachusetts Institute of Technology (MIT) license, which is open source and free for distribution.

### 4.3 Algorithms

Once the concepts related to the requirements of the model have been defined, the next step was established and define the implementation of the algorithms based on the concepts in Section 4.2. It is relevant to explain that some algorithms can handle more than one requirement. Also, there are specifications that do not necessarily require an equation or algorithm to be fulfilled, such as open-source and multi-platform applications.

The first step of the ARNI model is loading the data produced by a BCI device. To be able to do this, permission to read and write data from the workspace must be granted. Algorithm 1 presents the pseudo-code applied to load files with EEG data and its markers to be distributed by events. This process receives the OS path as a parameter to load the EEG files. The input for this algorithm will be one or more files that dwell in the OS directory received as a parameter. The next step is to identify the markers inside the EEG data and generate one or more output files in CSV format, which will be consumed in the next step. Further, the storage file format can be modified if required, in this way the model became more dynamic and embracing.

---

**Algorithm 1:** Load EEG data generated and split data based on markers.

---

```

input      : EEG data generated by a BCI device
output     : Multiple files based on markers
parameter: Path from the file system
1 begin
2   EegFileList  $\leftarrow$  FindEEGFiles (OSPath);
3   for  $i \leftarrow 1$  to GetLength (EegFileList) do
4     EEGData  $\leftarrow$  GetData (EegFileList ( $i$ ));
5     MarkerLogs  $\leftarrow$  GetMarkersLogs (OSPath);
6     for  $j \leftarrow 1$  to GetLength (MarkerLogs) do
7       MarkerPosList  $\leftarrow$  UpdMarkers (MarkerLogs ( $j$ ), EEGData);
8     end
9     MakeDirectoryIfNotExists (OSPath);
10    for  $k \leftarrow 1$  to GetLength (MarkerPosList) do
11      SubEEG  $\leftarrow$  GetEEGSubset (MarkerPosList ( $k$ ), EEGData);
12      SubsetFileName  $\leftarrow$  CreateFileName (MarkerPosList ( $k$ ));
13      SaveFileIntoOS (OSPath, SubsetFileName, SubEEG);
14    end
15  end
16 end

```

---

Once the markers have divided the EEG data, it begins the process to ready each event and applies the noise detection process with the pre-processing methods to clear the data collected. Algorithm 2 illustrates the process of loading subsets of EEG data and applying noise discovery and known pre-processing techniques in a pseudo-code structure. The OS path is used to locate the EEG files generated by Algorithm 1 as the parameter. Each event created by the tags is loaded separately, and it is submitted to high-pass and low-pass filters, as well as the DC offset method. Finally, the output is stored in the same path but with a different file name format, with cleared data that is ready to be processed by techniques such as FFT and ERD.

---

**Algorithm 2:** Noise detection and pre-processing actions.

---

```

input      : Multiple files based on markers
output     : Subset EEG files ready for neuroscience techniques
parameter: Path from the file system
1 begin
2   SubEegFilesList ← FindEEGFiles (OSPath) ;
3   EEGlab();          /* Startup Matlab Toolbox */
4   for  $i \leftarrow 1$  to length (SubEegFilesList) do
5     SubDataEEG ← importdata (SubEegFilesList ( $i$ )) ;
6     ParseFileName (SubEegFilesList ( $i$ )) ;
7     SubDataEEG ← RemoveColumnsNotUsed (SubDataEEG) ;
8     EEG ← importdata ('data',SubDataEEG,'srate',256) ;
9     EEG ← eeg_checkset (EEG);          /* Evaluate EEG */
10    EEG ← pop_chanedit (EEG,'load'); /* Channel Locations */
11    EEG ← pop_eegfiltnew (EEG, 2, 0); /* High-Pass */
12    EEG ← pop_eegfiltnew (EEG, 0, 30); /* Low-Pass */
13    MeanEEG ← GetMean (EEG.data, 2);
14    EEG.data ← EEG.data - repmat (MeanEEG, 1, length (EEG.data) );
15    EEG ← clean_rawdata (EEG, -1, -1, -1, 3, -1, 0.25);
16    SubsetFileName ← CreateFileName (SubEegFilesList ( $i$ )) ;
17    SaveFileIntoOS (OSPath, SubsetFileName, EEG.data) ;
18  end
19 end

```

---

The last step implements the functionality that will automatically measure the developer's level of comprehension. Algorithm 3 presents the methods and techniques used to translate the waves generated by a group of neurons into metrics. The analysis uses the wave frequencies applied to time-frequency analysis using FFT and ERD processes. The path from the OS is used as a parameter for Algorithm 3. The data used as input for this process is a subset of EEG files already prepared for data analysis and feature extraction. The output is a metric that will measure the program comprehension from the software developer's perspective based on the brainwave pattern. Several channels were used to calculate this metric as well as existing neuroscience methods.



---

**Algorithm 3:** Techniques proposed to measure program comprehension.

---

```

input      : Subset EEG files ready for neuroscience techniques
output     : Metric of program comprehension
parameter: Path from the file system
1 begin
2   ClearSubEegFilesList  $\leftarrow$  FindEEGFiles (OSPath);
3   LDC  $\leftarrow$  LoadDeveloperCharacteristics (ClearSubEegFilesList);
4   PSD  $\leftarrow$   $\emptyset$ ;
5   for  $i \leftarrow 1$  to Length (ClearSubEegFilesList) do
6     ReadDataFromChannels  $\leftarrow$  ClearSubEegFilesList ( $i$ );
7     for  $j \leftarrow 1$  to Length (ReadDataFromChannels) do
8       AlphaPeak  $\leftarrow$   $11.95 - 0.053 * \text{LDC.age}$ ;
9       Data  $\leftarrow$  ReadDataFromChannels ( $j$ );
10      while WindowEEGData  $> 0$  do
11        | PSD  $\leftarrow$  (FFT (NextSeg (Data), AlphaPeak, 125) + PSD) / 2;
12      end
13      if isWaveFromRelaxPhase (ReadDataFromChannels ( $j$ )) then
14        | RelaxPSD ( $j$ )  $\leftarrow$  PSD;
15      else
16        | TaskPSD ( $j$ )  $\leftarrow$  PSD;
17      end
18    end
19  end
20  RelaxBase  $\leftarrow$  GetMeanPSD (RelaxPSD);
21  for  $k \leftarrow 1$  to Length (TaskPSD) do
22    | Comprehension ( $k$ )  $\leftarrow$  (RelaxBase - TaskPSD ( $k$ )) / RelaxBase * 100;
23  end
24  SaveFileIntoOS (OSPath, Comprehension);
25 end

```

---

This section explains three different algorithms applied in the ARNI model. First, Algorithm 1 processes the data generated by an EEG device while a software developer is reading and interpreting the source code. Second, Algorithm 2 illustrates the noise detection that is being processed and the various pre-processing techniques such as high-pass and low-pass. Third, Algorithm 3 shows the methods used to process the wave pattern and source to measure the program compression from a programmer. Section 4.4 presents the technique developed to support the ARNI model and the cognitive effort calculation.

#### 4.4 Cognitive Effort Technique

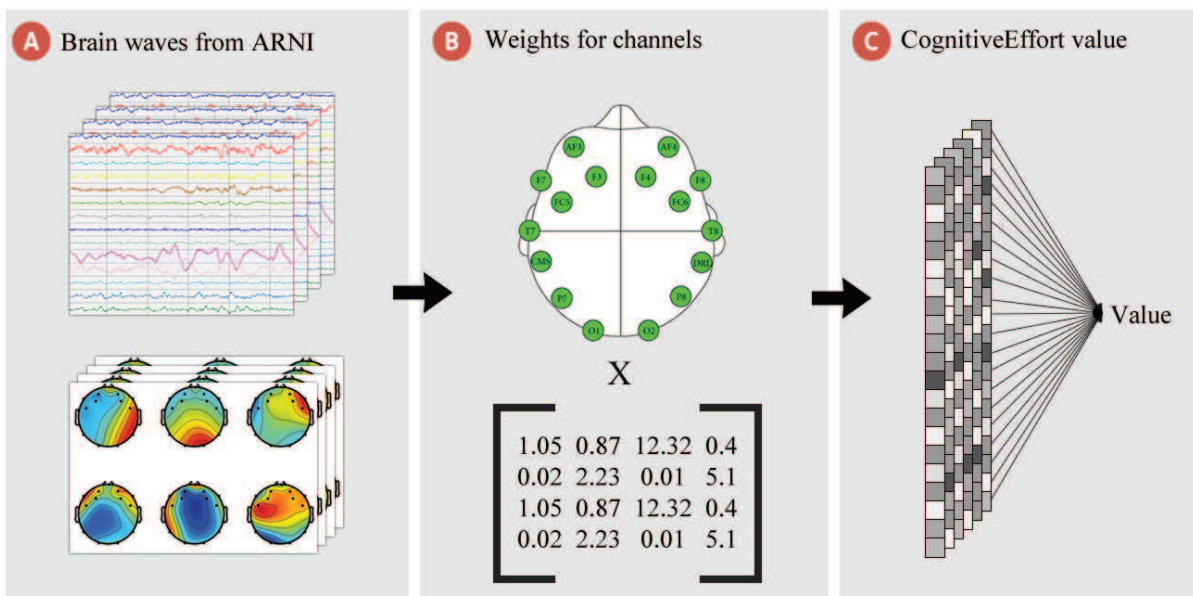
The algorithms detailed in the last section focused on load, clean, and processing the raw EEG data. Once all these steps are complete, the last task is to calculate the cognitive effort using the *CognitiveEffort* technique. The *CognitiveEffort* is a technique that mainly uses ERD

calculated for each channel in the brain and applies different weights based on area and its priority to calculate the effort for a developer. Equation 4.1 shows the inputs that are required to generate the cognitive effort during a specific task. The calculation uses the  $\varphi$  as in input for waves processed through the ARNI model for as many channels as the EEG device has. For each  $\varphi$ , weight ( $W$ ) is associated based on the area in the brain and the type of activity that is being executed. Once the execution of the equation is completed, a value that estimates the cognitive effort during the program comprehension task is measured.

$$CognitiveEffort = \frac{\sum_{i=1}^n W_i \times \varphi_i}{\sum_{i=1}^n W_i} \quad (4.1)$$

The process to calculate the *CognitiveEffort* is illustrated in Figure 21. First, Figure 21A shows the brain waves collected from the ARNI model. Each channel generates an ERD value as output. Second, Figure 21B presents the channel relation from the EEG device, based on the 10/20 international standard, and also the weights associated with each channel. Third, Figure 21C details the collection of values calculated, based on the weight, in order to generate the final result. The final results represent the cognitive effort generated by a software developer to comprehend a specific section of source code based on their brain behavior.

Figure 21 – Process of CognitiveEffort technique.



Source: Created by the author

Focusing on the explanation outlined above, Figure 22 exemplifies the step-by-step mathematical calculation. This example simulates an EEG device with four channels and using the



weights (0.75, 0.5, 1.25, 1) for the channels AF3, F7, F5, and FC3, respectively. Figure 22A illustrates fundamentally the technique produced, which will interact four times, once for each channel. Figure 22B details all values extracted for each channel (-23.542, 32.53, -2.32, 43.23) multiplied by the respective weight. Figure 22C presents the values (-17.6565 + 16.265 - 2.9 + 43.23) calculated based on the ERD and the weights defined, divided by the sum of the weights (3.5). Finally, Figure 22D shows the final value calculated for the *CognitiveEffort* technique that completes the ARNI model proposed in this research.

Figure 22 – Calculation example of CognitiveEffort.

<p>A)</p> $CognitiveEffort = \frac{\sum_{i=1}^n W_i \times \varphi_i}{\sum_{i=1}^n W_i}$	<p>B)</p> $CognitiveEffort = \frac{\sum_{i=1}^4 [(0.75 \times -23.542), (0.5 \times 32.53), (1.25 \times -2.32), (1 \times 43.23)]}{\sum_{i=1}^4 [0.75, 0.5, 1.25, 1]}$
<p>C)</p> $CognitiveEffort = \frac{-17.6565 + 16.265 - 2.9 + 43.23}{3.50}$	<p>D)</p> $CognitiveEffort = 11.1252857$

Source: Created by the author

This section explains the *CognitiveEffort* equation used as the final step in the ARNI model. Additionally, the equation uses specific weights for each channel, and it allows a high degree of customization based on the context applied, such as program comprehension. Section 4.5 presents the architecture of the ARNI model and the compositions of the modules.

#### 4.5 Architecture

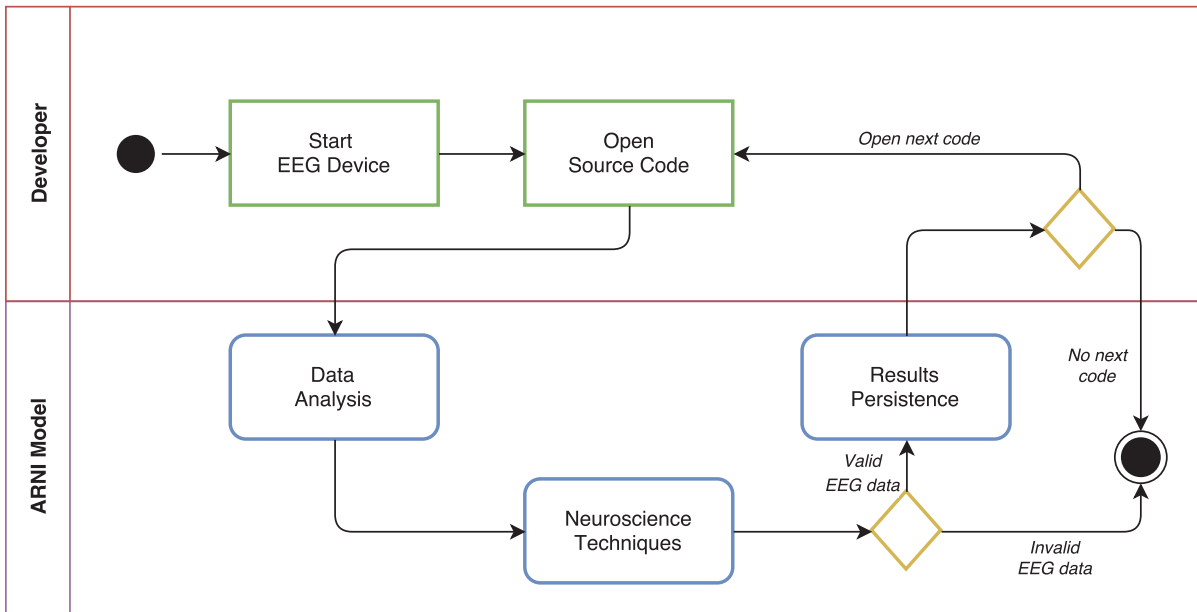
This section will present and address the architecture used in the ARNI Model. The development uses an extensible architecture to accommodate an adaptive process for multiple EEG devices and source code technologies. Hence, the developer might add new components or even change functionality inside the ARNI model. Three modules were developed to support the ARNI model. The first module is data analysis, which is responsible for processing the markers and dividing the data based on the event created by the software developer. Second, the pre-processing techniques in neuroscience, such as filter, FFT, and ERD are applied in the EEG

wave. Third, the persistence module has the function of storing the results in the workspace and choosing to run the measure of the program comprehension process.

Figure 23 presents an overview of the execution process using the ARNI model. Initially, the programmer must turn on the EEG device and wear it connected to an OS. Next, a marker is associated with the snipped code displayed on the screen. Following this, the data analysis will split the EEF data collected by the marker. Neuroscience techniques will then be performed to clean and process the information gathered.

Once complete, if the results are not acceptable, such as noise interference, the process marks that part as invalid. However, if the data is valid, the results will be written to disk in the workspace used by the software developer. In case there are more source code snippets to display, the whole process will continually repeat until there is no further code to display, at which time the process terminates. Upon termination, the data is stored in a file-based format for further analysis if required.

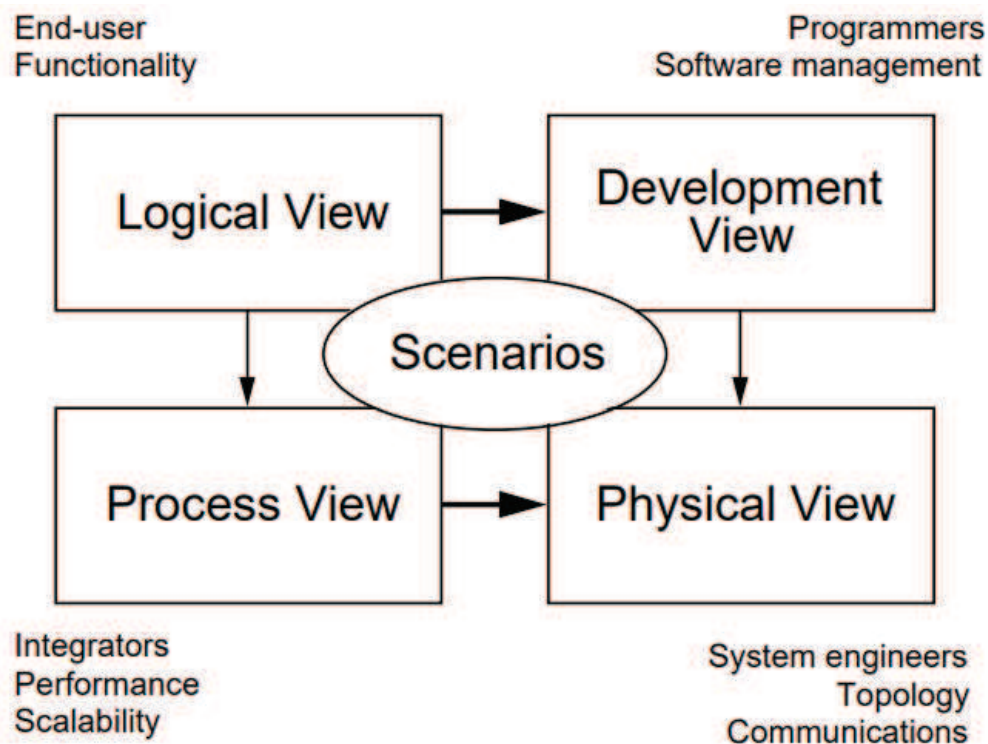
Figure 23 – Developer overview of the execution process.



Source: Created by the author

Figure 24 presents the methodology "4+1" architectural view used to describe the ARNI model (KRUCHTEN, 1995b). It proposes four perspectives to visualize the system: logical view, process view, development view and physical view. The logical view is represented by the features model in Subsection 4.5.1 The process view is presented as an activity diagram in Subsection 4.5.2. The component diagram is used to illustrate the development view in Subsection 4.5.3. Layers from the ARNI model are displayed in Subsection 4.5.4 using a diagram.

Figure 24 – The methodology "4+1" view architecture.



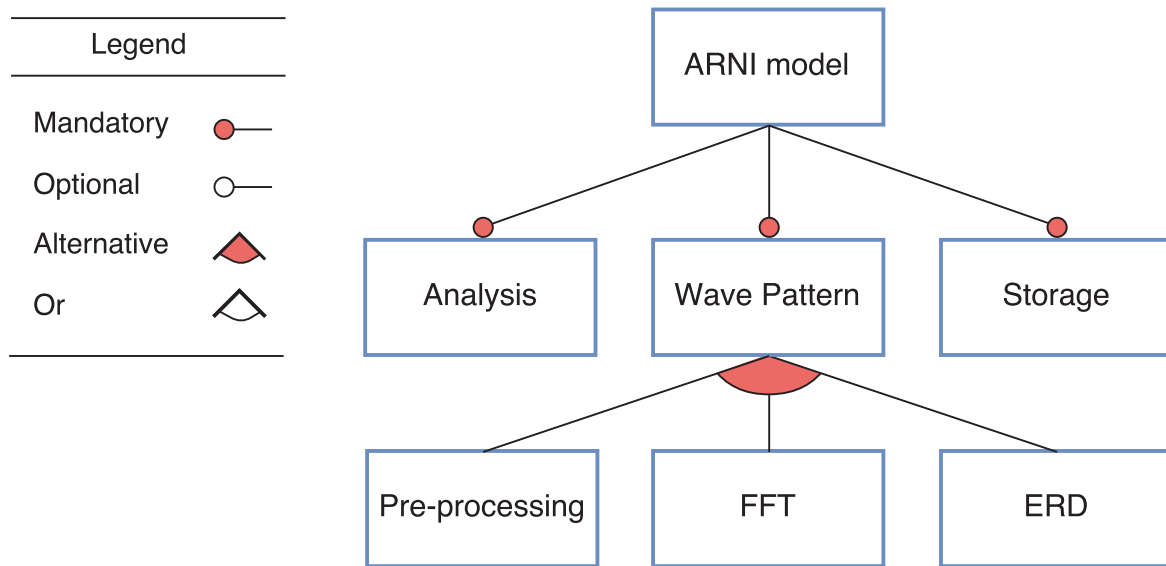
Source: Kruchten (1995a)

#### 4.5.1 Features Model

Feature models are defined as a condensed representation of the characteristics of a product. These features can create various combinations of applications from the same software. There are two reasons why feature models were chosen to represent the logical view. First, they detail from the software perspective the derivations and points of variability. Second, feature models have been used in previous works to represent similar perspective of software (CLARKE, 2001; OLIVEIRA; BREITMAN; OLIVEIRA, 2009; FARIAS et al., 2015c). Finally, the architecture for the ARNI model provides the fundamental properties to measure program comprehension using EEG data generated by a BCI device from a programmer.

Figure 25 displays a compact representation of the features of the ARNI model. There are three necessary characteristics in the ARNI model: analysis, wave pattern, and persistence in obtaining the results. Attention was paid throughout the ARNI architecture to guarantee that the process complies with the cognitive effort measure through EEG data. Thus, the decisions made provide a set of key features, marker discovery in EEG data, noise detection, wave pre-processing, FFT process, and ERD measurement. Therefore, for the software developer to be able to use the ARNI model, they must implement the required characteristics. At least one wave pattern must be applied in order to process the EEG data.

Figure 25 – Features diagram with components of ARNI model.



Source: Created by the author

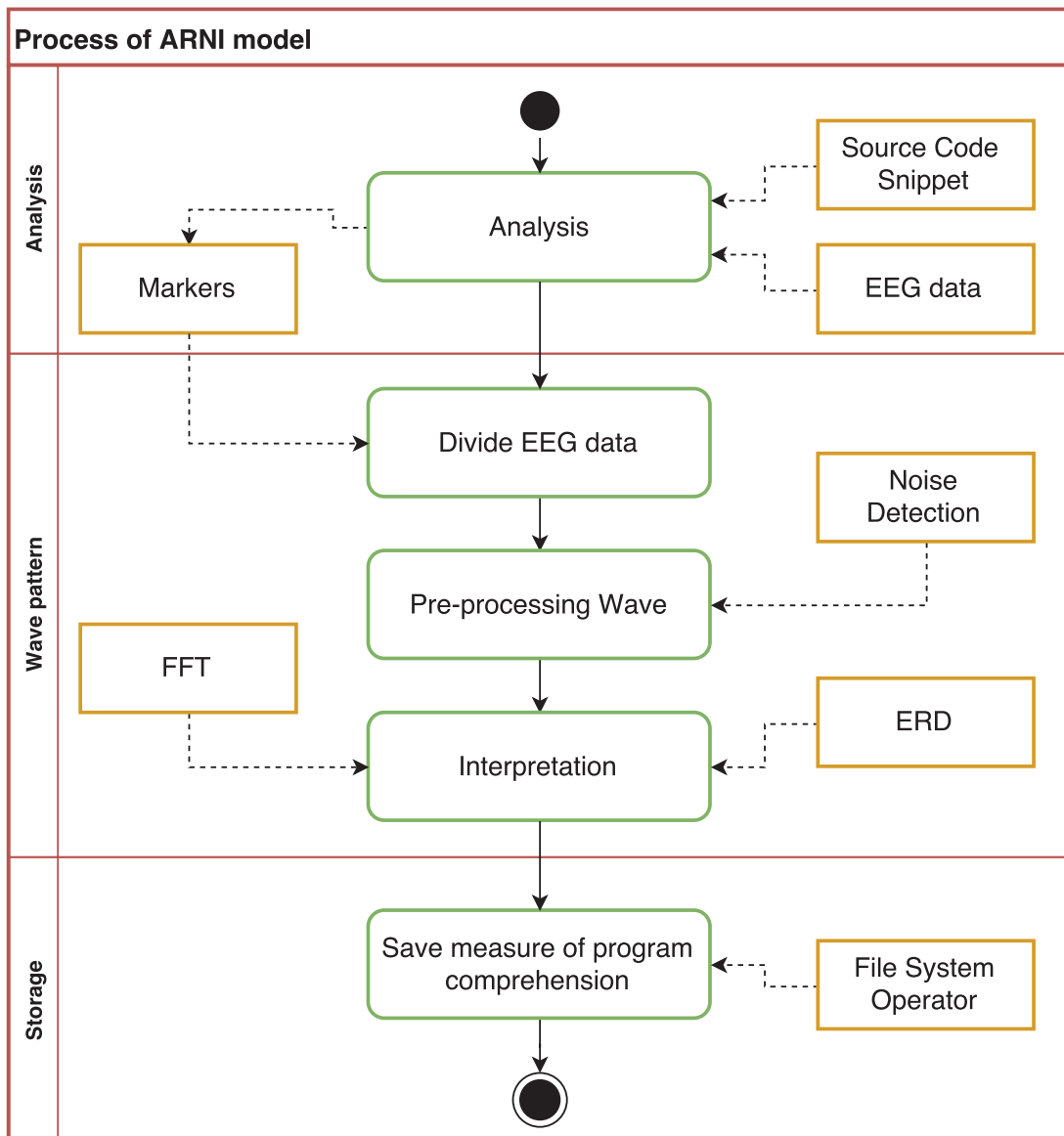
#### 4.5.2 Activity Diagram

The ARNI process will be described in this subsection using an activity diagram. This type of diagram was chosen to represent the process view of the "4+1" architecture because it describes aspects of the system as well as activity interaction between steps. Figure 26 details the process of EEG data translation into a metric to measure program comprehension. Overall, it includes the artifacts, activities, and results concerning the ARNI model.

**Analysis phase** The goal of this step is to validate the inputs and check if they are synchronized for further steps. The first step is to validate if the BCI device is properly synchronized for wireless technologies such as Wi-Fi or Bluetooth. The next step is to display the source code that the programmer will need to understand, which can be various scenarios such as code review or debugging. Additionally, a marker is connected with each part of the source code and then integrated directly into the data generated by the BCI device.

**Wave pattern phase** The main objective of the wave pattern phase is to measure program comprehension relying on BCI information. There are two inputs for this phase: EEG data and markers. After input, three steps are executed: divide EEG data, pre-processing wave, and interpretation. First, the EEG data is parsed using the channels logged and the markers inserted. If a gap in the data is found, a calculation for that part is done based on the sample rate from the BCI device. Second, the EEG data divided is then analyzed for noise detection such as the blink of an eye or any strong movements. After the data is cleaned, the frequencies theta and alpha are extracted since they are most likely to determine cognitive information. The defined wave strategy is broken down into smaller segments, which are then run through the FFT technique

Figure 26 – Process of ARNI model for program comprehension.



Source: Created by the author

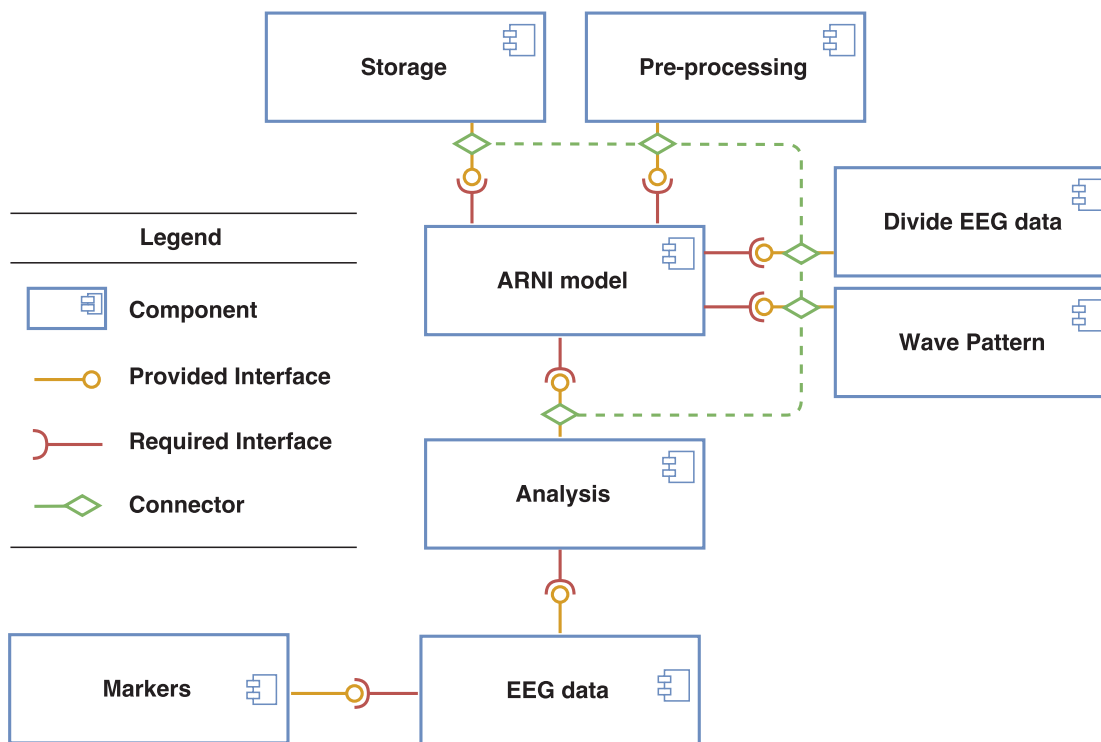
for PSD extraction. Once the PSD is calculated for the rest baseline and the task event, the next step is to calculate the ERP for the source code snippet. The result expected from this phase is a matrix of measurement of program compression for each source code segment.

**Storage phase** The intention of this phase is to store the results obtained from the brainwave analysis. The results will be stored on a disk or integrated with other APIs such as development IDEs. Once the results are saved, the measurement can be used with other techniques to help others perform analyses using the ARNI results as input. Operation-based storage is a method that saves this kind of measurement as meta-data with the source code itself, such as SVN.

4.5.3 Component Diagram

A component diagram is known for describing the elements in a physical view. The components can be modularized in elements to allow for its' reuse as well as to allow for timely maintenance. Based on the features model in Figure 25, each element is represented as a component. Figure 27 presents each part encapsulating its behavior through a set of internal activities, which behave as an independent module. The ARNI model component is the main engine of the model and focuses on managing all other processes. Also, it includes initiating the execution of activities, storing the progress, and evaluating the return of the given technique.

Figure 27 – Component diagram of ARNI model.



Source: Created by the author

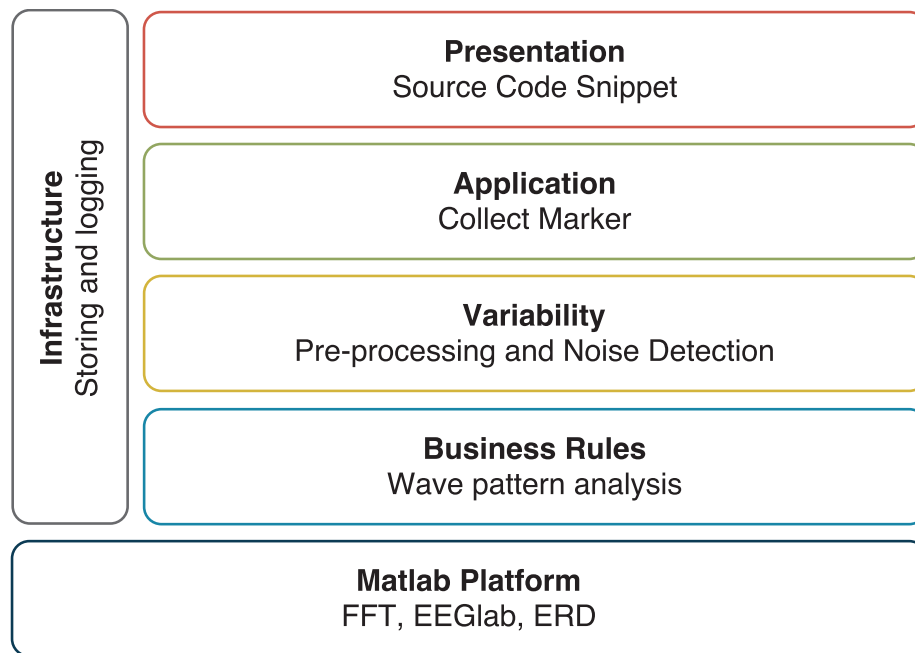
The marker component, seen as a block like the others, is responsible for sending the marking code in a numeric format within a serial link. The EEG data added these markers in the next entry based on the frequency of the BCI device. The analysis component checks for data integrity and when ready, advances it to the process. Once obtained, the data is then submitted to a divide process, which will read the data and break it down by event level. The filter is executed in the data to isolate the essential frequencies. Techniques, such as FFT and ERD are then performed through the data to calculate the program comprehension. In the storage layer, it stores the outcomes produced by the models during the interpretation by a software developer. The data is then saved in a text file on the hard disk.

#### 4.5.4 Multilayer Architecture

Multilayer architecture provides envisioning the structure from a development perspective. The layer organizes cross-cutting interests individually by the features diagram. As shown in Figure 28, five layers organize the architecture developed. The presentation layer represents the source code and receives the input data based on the segment presented, which is required along with the EEG data.

The application layer is the same as the ARNI model engine. It orchestrates the wave pattern recognition and executes other components in the model. The variability layer is responsible for implementing noise detection that is in charge of interweaving the clear data and external stimulus based on the developer's behaviors. The business rule layer accommodates algorithms that measure program comprehension through a cognitive load process. Finally, the infrastructure layer handles data access, exceptions, storage, and logs.

Figure 28 – Layer of ARNI model.



Source: Created by the author

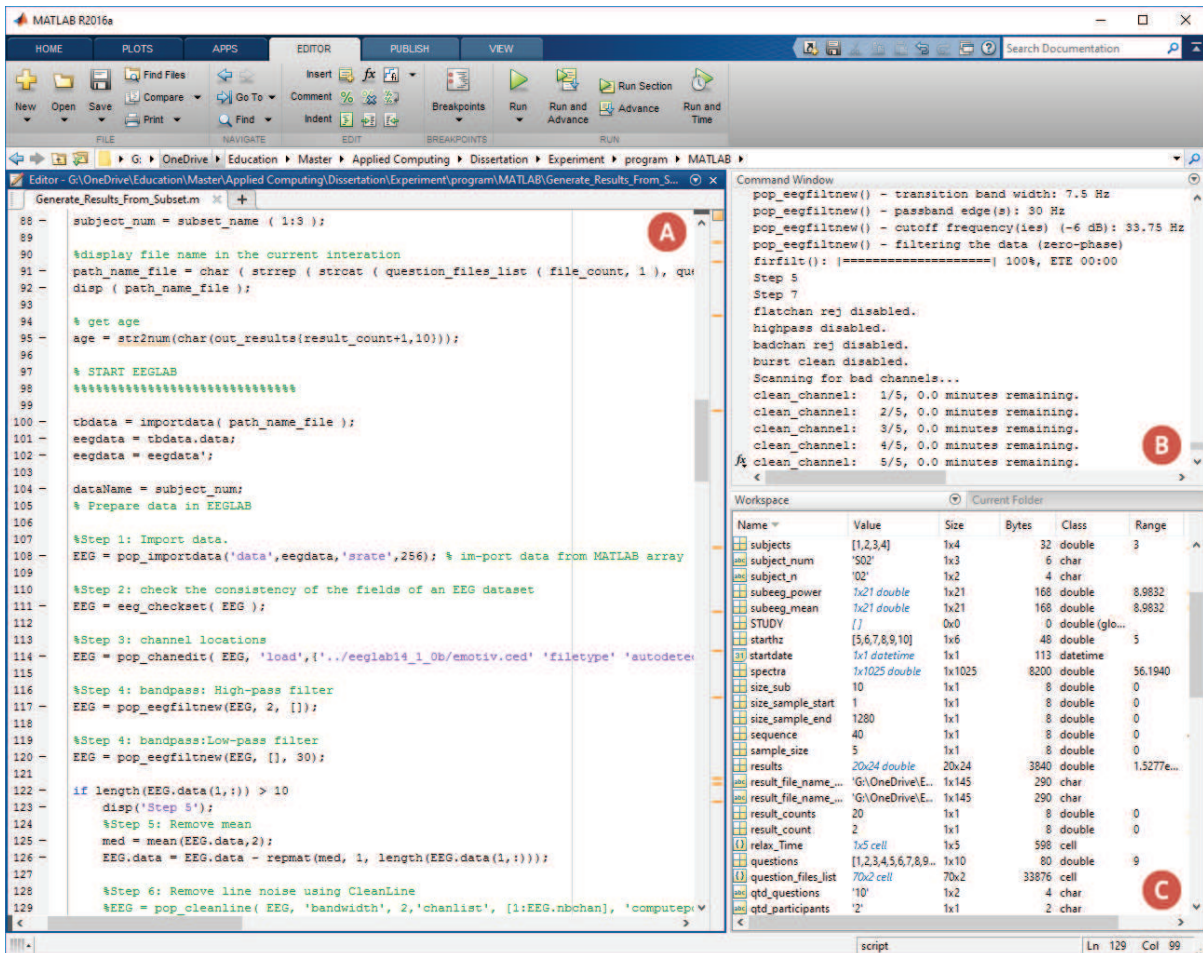
#### 4.6 Aspects of Implementation

This section presents the prototype created with the purpose of evaluating the ARNI model proposed, as well as displays the process to measure program comprehension. The prototype implemented uses the Matlab platform as support for mathematical calculation and, at the same time, allows the model to support multi-platform technology as presented in Figure 29. Matlab



provides several toolboxes used in this study. For instance, the open-source EEGLab toolkit allows pre-processing and noise detection methods. The minimum number of external libraries was used to process the data to test the prototype as generically as possible. The prototype reads and filters the files generated by the BCI device with the markers and then executes the methods to measure program comprehension.

Figure 29 – ARNI model prototype.



Source: Created by the author

Figure 29 shows an overview of the prototype with three uppercase letters from A to C representing the elements. Figure 29A represents the code for loading the EEG data in a raw format into the system. In the editor, the transformations are made by using the toolkits from MATLAB, as well as the language of this platform to transform wave behaviors. Figure 29B illustrates the output of the data generated. The steps of the process can be visualized, and actions can be requested if some undesired behavior occurs. Figure 29C shows the workspace values and the values of the measurement of program comprehension. Even though the results are recorded in the OS in a text arrangement, this space also allows the integration of memory between APIs, which can reduce time when integrated with other applications.



#### 4.6.1 Technologies

The developed prototype based on the ARNI model uses MATLAB programming language in its multi-platform IDE. This IDE enables a software developer to extend functionality based on the toolkit technology. This platform was chosen as it is optimized for scientific problems which allows for executing very complex calculations in a fraction of the time. The prototype presents the following technologies and tools to collect the EEG data, mark it, process, and generate results based on the following technologies.

**Emotiv Xavier TestBench (Version 3.1.21)** The Emotiv Xavier TestBench is an application which runs as a stand-alone program. TestBench autonomously collects data packets from the universal serial bus (USB) device and it stores the EEG signals in an OS. The EPOC+ neuroheadset model converts the data to digital form, which becomes ready for further brainwaves processed. The data is wirelessly transmitted to the USB receivers. Emotiv Xavier TestBench panel is a status pane, that evaluates the sensor contact quality before starting recording. The TestBench application creates files with EEG data in the OS in the EDF format, which can later be converted into CSV. Markers tunnel is used to specify events in EEG files as they are being stored and which can be used later during data analysis.

**Paradigm Stimulus for Desktop (Version 2.5)** The Paradigm is a standalone application used to display content in a desktop environment. It allows the user to create different types of stimulus and connect them with external protocols. The layout of the presenting information is created using the stimulus designer with instructional events. It allows displaying text such as a source code snippet. The main reason to use this application to handle image manipulation on the screen is that it has a millisecond precision and can handle markers, which essentially controls all events that occur in the environment.

**Paradigm Element For Ports (Version 1.6)** The paradigm element for ports is an extension of the Paradigm Stimulus application. It has a main functionality that establishes communication with EEG devices and external devices. This extension enables the use of device ports to send triggers to the Paradigm's device manager module. An assigned value is then attributed with each event as it occurs within the application. The major reason for using it concentrates in the fact that markers can be assigned to images in a millisecond precision level, and after that, send it through the network.

**Virtual Serial Ports (Version 3.22)** The Virtual Serial Ports is a software used to imitate physical serial ports in a virtual context. It acts exactly like the hardware port functionality, implementing the configuration of data bits. This application emulates custom plug-and-play (PnP) serial ports, such as COM as well as others. The network in from software is essential to virtual connect port COM1 and COM2 as a redirector. This connection makes Paradigm, and Emotive TestBench communicate with each other using 19,200 bits per second with an 8-bit data size. Once the software is running in the background, this software becomes mainly the controller of all markers injected in the EEG data generated by the EEG device.

**MATLAB R2016a (Version 9.0)** The Matrix Laboratory (MATLAB) is a platform which uses a numerical computing environment as a paradigm for development. It is widely known due to its vast library of toolboxes with algorithms essential to several domains. The math calculations using this platform execute much faster as the data is loaded in memory before the calculation. This platform was selected as it is a multi-platform and runs in the major OSs. MATLAB can handle gigabytes of EEG data that is generated in a matter of seconds since performance is crucial in the neuroscience field.

**EEGLab Toolbox (Version 14.1.0b)** The EEGLAB is a MATLAB toolbox for processing event-related EEG's that uses the powerful MATLAB scripting language as the foundation of its features. The EEGLAB works in an interactive process with high-density EEG and correlated data, such as events, channel locations, and epoch information. The EEGLAB toolbox implements methods for visualizing and analyzing event-related brain dynamics and visualizing event-related EEG's. This toolbox has been selected for a number of reasons, but mainly as it centralizes the best techniques to work with brainwave data in one place.

## 5 EVALUATION AND RESULTS

This chapter details the evaluation conducted in regards to a controlled experiment, which results in empirical results. In particular, empirical studies are crucial in endorsing suggested concepts and examining the effects of the produced ARNI model. A lack of empirical studies can be found in the program comprehension field regarding the cognitive effort and BCI.

Primarily, a controlled experiment was designed to evaluate and analyze the data generated by different degrees of complexity in the source code. This experiment aimed to evaluate the effects of modularity in the cognitive effort using the developed model in this study. In practice, five questions involving two opposite categories were designed to measure human factors when different source code snippets are being reviewed and understood by software developers.

In the experiment, the ARNI was the method used to support this study. This evaluation follows the recommendations found in (WOHLIN et al., 2012), as well as based on previously published works (FARIAS; GARCIA; LUCENA, 2014; FARIAS et al., 2015b). Also, several precautions were used to reduce the side effects of non-invasive EEG devices. First, two pilot questions were added to the experiment to ensure that each of the 35 volunteer software developers understood the experimental process, generating 350 trials in total. Two control questions were added to validate that the subjects actually understood the source code.

Substance restrictions were added for the participants to reduce the variance of brain activity by disallowing the use of caffeine, alcohol, or legal/illegal drugs. In the end, questionnaires were used to collect subject information and the participants' impressions of the experiment. Statistical tests were then performed and their outcomes compared against the results generated by the ARNI model. After that, assumptions were made on the basis of these results and which will guide future studies.

The overview of this chapter proceeds in six sections as follows: Section 5.1 presents the objectives, hypotheses, and the context in which the experiment was intended to evaluate the ARNI model. Also, it focuses on the activities and scenarios carried out during the experiment. Section 5.2 details the organization of the controlled experiment. Section 5.3 will detail the results obtained from the experiment. Section 5.4 will discuss the findings and implications of program comprehension. And finally, section 5.5 emphasizes the limitations involved in this evaluation and the methods used to mitigate them.

### 5.1 Experimental Drawing

This section addresses the process in the controlled experiment, such as the objectives, hypothesis and variables used to evaluate the ARNI model. Nevertheless, little is known about the degree of modularity and its effect in the cognitive process (HANNEMANN; KICZALES, 2002). Some factors suggest various barriers to implementing modularity at an acceptable degree of effectiveness (DIETRICH et al., 2010). The high-level idea of the controlled experiment

is to be able to evaluate the cognitive effect of modularity code in developers. ARNI was evaluated through a controlled experiment, in which the effects of modularization on cognitive effort were investigated.

Code that goes against the rules, sometimes classified as bad smells, is usually the problem in source code maintenance, and again, modularity is advised as such (MUNRO, 2005; MOHA et al., 2010). In addition, the cognitive process plays an important role during development and maintenance processes, and knowing when the source code is under-modularized or over-modularized becomes even more relevant. While this experiment was planned, a consent letter was requested from the coordinator of PPGCA to allow the running of the experiment using EEG devices as presented in Annex B. The design of the experiment is defined in three subsections. Subsection 5.1.1 presents the objectives related to the controlled experiments using the GQM format. Subsection 5.1.2 details the hypotheses created to achieve the objectives. Subsection 5.1.3 shows the activities and scenarios built for the execution of the experiment.

### 5.1.1 Objectives

This subsection formalizes the objectives of the experiment. It explores the effects of cognitive effort through the developers' understanding of the program during the source code comprehension exercise. Additionally, the experiment analyzes the degrees of comprehension from the developers' point of view, concerning their respective levels of understanding. These effects are investigated involving ten scenarios with different levels of complexity concerning comprehension tasks. Activities of understanding regarding effort and correctness are also evaluated. The purpose of this investigation is presented in the GQM (Goal Question Method) conventional (WOHLIN et al., 2012).

**Analyze levels of modularization  
for the purpose of investigating their effects  
with respect to cognitive effort, correctness, and temporal effort  
from the perspective of software developers  
in the context of program comprehension.**

The proposed experiment evaluated the effect of modularity, complexity and the effort employed to understand the source code behavior. The experiment is divided into two main groups: non-modularity and modularity. Five questions are employed in each of the modules (ten questions in total). The experiment measures the cognitive effort using the ARNI model and different degrees of modularity and complexity described in Section 1.4. The objective will be used to define the hypotheses of the experiment. Note that the level of knowledge or experience is another factor that can affect the comprehension of source code (RICCA et al., 2010). The degree of knowledge, as well as experience, is taken into consideration and examined.

### 5.1.2 Hypotheses

According to the experiment goals of this study, it aims to measure the cognitive effort, timing, and correctness of the program comprehension in a non-modular and modular manner. Based on these goals, three distinct hypotheses evaluate the ARNI model developed in Chapter 4. First, Hypothesis 1 analyzes the temporal effort to understand and give an answer to the source code displayed. Hypothesis 2 evaluates the correctness of the responses by the code interpretation. Finally, Hypothesis 3 validates the cognitive effort from the rest moment until the developer has a complete comprehension of the source code snippet.

**Hypothesis 1: Temporal Effort** The action of understanding modular source code has a tendency of simplicity. Nevertheless, the possibility of multiple levels and combinations become costly when involving several degrees of methods connecting with one another. The hypothesis evaluates whether the time consumed to understand and the metric generated by the prototype can be measuring cognitive process. Based on these assumptions, the null and alternative hypotheses of the program comprehension effort (PCE) are presented below:

**Null Hypothesis 1,  $H_{1-0}$ :** Modular source code requires more or the same temporal effort than non-modular source code.

$$H_{1-0} : \text{Temporal Effort}(\text{code})_{\text{Non-Modularity}} \geq \text{Temporal Effort}(\text{code})_{\text{Modularity}}$$

**Alternative Hypothesis 1,  $H_{1-1}$ :** Modular source code requires less temporal effort than non-modular source code.

$$H_{1-1} : \text{Temporal Effort}(\text{code})_{\text{Non-Modularity}} < \text{Temporal Effort}(\text{code})_{\text{Modularity}}$$

**Hypothesis 2: Correctness** Inconsistencies affect the misinterpretation of the source code due to mental model conflicts for the understanding of the source code segment. The presence might influence the correct way in thinking about the coding process or not based on the contexts created by modularity. This hypothesis evaluates whether the complexity, as well as modularity, may affect the developers thoughts and efforts using the prototype to analyze the effects. It is necessary to assess the correct output from the developers' context and measure the effort during this task as presented below:

**Null Hypothesis 2,  $H_{2-0}$ :** Modular code increases or keeps the rate of correct interpretations when compared to non-modular code.

$$H_{2-0} : \text{Correctness}(\text{code})_{\text{Non-Modularity}} \leq \text{Correctness}(\text{code})_{\text{Modularity}}$$

**Alternative Hypothesis 2,  $H_{2-1}$ :** Modular code reduces the rate of correct interpretations when compared to non-modular code.

$$H_{2-1} : \text{Correctness}(\text{code})_{\text{Non-Modularity}} > \text{Correctness}(\text{code})_{\text{Modularity}}$$

**Hypothesis 3: Cognitive Effort** Cognitive Effort plays an important part in a maintenance task while the developer is trying to understand what is happening with the code. It becomes vital to validate the cognitive effort when a similar source code with various complexities can contribute by modularity. This hypothesis evaluates whether the non-modularity source code and modularity may affect the developers thoughts and understanding the source code to analyze the effects. Based on the points above, the null and alternative hypotheses to measure program comprehension effort (PCE) are presented below:

**Null Hypothesis 3,  $H_{3-0}$ :** Modular code reduces or equalizes the cognitive effort from the brain when compared to non-modular code.

$$\mathbf{H_{3-0}} : \text{CognitiveEffort}(code)_{Non-Modularity} \leq \text{CognitiveEffort}(code)_{Modularity}$$

**Alternative Hypothesis 3,  $H_{3-1}$ :** Modular code increases the cognitive effort from the brain when compared to non-modular code.

$$\mathbf{H_{3-1}} : \text{CognitiveEffort}(code)_{Non-Modularity} > \text{CognitiveEffort}(code)_{Modularity}$$

This subsection provided an overview of the null and alternative hypotheses in three different scenarios of this controlled experiment. These hypotheses seek adequate support to answer the research question of the study. The hypothesis intends to validate the ARNI model and the prototype developed. The measurement of program comprehension using the cognitive effort is the main goal produced in this study.

### 5.1.3 Variables

This section discusses the dependent and independent variables involved in the experiment. This research identifies variables that represent increasing levels of complexity, or modularization, within non-modularity and modularity groups. With this in mind, the analysis aims to measure the program comprehension effort level while at the same time understanding the source code. The effectiveness and relevance of background knowledge might raise different levels of comprehension while comparing fewer experienced subjects. Table 13 details the variables investigated in this research with the goal of validating the proposed model.

This controlled experiment is composed of three dependent variables: temporal effort (TE), correct comprehension rate (CCR), and cognitive effort (CE) calculated by ARNI. The EA identifies the temporal effort used to detect an answer, in other words, to process the information and define a solution, even with the correct answer. Defined in seconds, the TE measures the period between displaying a question and inputting an answer within the 60 second limit. The CCR has the goal of measuring the degree of assertiveness answered correctly for the program comprehension tasks. The CCR calculates the ratio of the sum of the correct answer, divided by the total amount of tasks answered, using the formula  $CCR = \text{correct answers}/\text{total answers}$ .

Lastly, the CE represents the degree of cognitive effort used to reply to a task. The CE calculates the ratio of the brain synapses happening in the work area of the developer's brain. The CE value is produced by using the ARNI model proposed in the earlier Chapter 4. This value is adapted per individual based on the IAF. It indicates the total power needed using the IAF algorithm to find process for the program comprehension.

Table 13 – Variables used in the controlled experiment.

Type	Variable name	Scale
Independent	Abstraction Degree (AD)	Nominal: Non-Modularity, Modularity
	Level of Complexity (LC)	Nominal: 1 (Low), 2, 3, 4, 5 (High)
Dependent	Temporal Effort (TE)	Interval [0 .. 60]
	Correct Comprehension Rate (CCR)	Interval [0 .. 1]
	Cognitive Effort (CE)	Interval [-200 .. 100]

Source: Created by the author

#### 5.1.4 Scenarios

A controlled experiment was performed to measure the temporal effort, correctness, and cognitive effort of the program comprehension. This subsection presents the scenarios applied during the experiment, as well as the details involved before, during, and after its execution. Each task is composed of two scenarios, modular and non-modular. All scenarios evaluated in the experiment were randomized to reduce sequence influence in any way.

The independent variables are divided into two distinct groups: the abstraction degree (AD) and the level of complexity (LC). The AD has the goal of splitting the task into two main groups: non-modularity and modularity. Each group is composed of five distinct tasks, using the JAVA programming language for the questions. For each task in the one group, there will be a corresponding task in the opposite group, which will be similar in its structure and answers. The LC is specified to make each task progressively more difficult. The first task is the simplest task, and thereafter, each task in both groups will have a higher level of difficulty. The most difficult task in both groups are labeled as Task 5. The LC is used mainly to encourage the developer to resolve similar tasks, but it will have different levels of complexity.

During the experiment's execution, the first two questions were added as a pilot to reduce the influences of knowing the execution process. These questions were not included in the randomized process. Two additional scenarios were inserted in the middle of the experiment to control the validity of the answers given during the experiment. The goal of the first scenario is



validating if the developer understands how the JAVA language works in its fundamentals. The second control scenario was used to see if the developer was guessing or correctly interpreting the sentence displayed.

Table 14 presents ten evaluation scenarios which are divided into two main groups: non-modular and modular. The set *NM* identifies each scenario for non-modular and *m* for modular, concatenated with the letter T for the task and the level of difficulty aggregated. For instance: NMT1; NMT2 ... MT4; MT5; where NM indicates the scenario without modularity at the beginning of the identifier and M scenarios with modularity applied. Each group has its task classified by the task numbers from 1 to 5. Each level presents an incremental cognitive complexity to process the data and find an answer without additional support. A specialist in software development defined the order, which mainly follows the HE measure, except for one case.

Table 14 – Scenarios of the controlled experiment.

Group	Label	ID	LD	C	M	I	LOC	MCC	JS	HE	HL	MI
Non-Modular	Task 1	NMT1	1	1	1	0	10	1	8	2542	49	110
	Task 2	NMT2	2	1	1	0	8	1	6	2187	51	115
	Task 3	NMT3	3	1	1	0	4	1	4	737	32	125
	Task 4	NMT4	4	1	1	1	10	2	10	3319	63	105
	Task 5	NMT5	5	1	1	1	19	3	22	8715	100	89
Modular	Task 1	MT1	1	1	5	0	18	5	12	1150	81	141
	Task 2	MT2	2	2	2	0	14	2	10	2398	72	120
	Task 3	MT3	3	5	5	0	30	5	20	2908	120	126
	Task 4	MT4	4	2	2	1	16	3	14	3932	85	113
	Task 5	MT5	5	5	5	1	43	7	38	13433	192	112

**Legend:** (*LD*) level of difficulty, (*C*) classes, (*M*) methods, (*I*) interactions, (*LOC*) lines of code, (*MCC*) McCabe cyclomatic complexity, (*JS*) Java statements, (*HE*) Halstead effort, (*HL*) Halstead length, (*MI*) maintainability index

Source: Created by the author

The scenarios listed in Table 14 detail the description of the elements contained in each task. A few characteristics were extracted from each scenario with the assistance of the JHawk 6.1 (Virtual Machinery, 2017). The list starts with the level of difficulty (LD), number of classes (C), number of methods (M) and iterations (I), the total lines of code (LOC), as well as the total number of Java statements.



Four known metrics were used to extract even more information: McCabe cyclomatic complexity (MCC), Halstead effort (HE), Halstead length (HL), and maintainability index (MI). The MCC indicates the complexity of a program as well as measuring the number of linearly independent ways throughout a source code (MCCABE, 1976). The HE and HL measures the mental effort required and the program length, which is computed statically from the program's source code (CURTIS et al., 1979). The MI is calculated as a factored formula composed by the TL, HV and MCC (WELKER, 2001). Figure 30 illustrates the MT5 task used by the subjects during the experiment execution.

Figure 30 – Task 5 - MT5 - modular scenario.

```

public class Cards {
    public static void main(String[] args) {
        int[] tasks = { 4, 2, 7, 6, 3, 2 };
        int qtd = 0;
        for ( int n = 0; n < tasks.length; n++ ) {
            switch ( tasks[n] ) {
                case 2: TypeA a = new TypeA();
                       qtd = a.plus ( qtd );
                       break;
                case 4: TypeB b = new TypeB();
                       qtd = b.plus ( qtd );
                       break;
                case 6: TypeC c = new TypeC();
                       qtd = c.plus ( qtd );
                       break;
                default: TypeD d = new TypeD();
                        qtd = d.plus ( qtd );
                        break;
            }
        }
        System.out.print( qtd );
    }
}

public class TypeA {
    public int plus ( int vl ) {
        return vl + 2;
    }
}

public class TypeB {
    public int plus ( int op ) {
        return op + 5;
    }
}

public class TypeC {
    public int plus ( int fy ) {
        return fy + 3;
    }
}

public class TypeD {
    public int plus ( int ts ) {
        return ts + 1;
    }
}

```

1) 13      2) 10      3) 12      4) 11      5) 14

Source: Created by the author

Appendix F details the non-modularized questions in the experiment. Each of these have a specific LD from very low, labeled as 1, to very high, labeled as 5. Appendix G presents the questions focused on the modularity portion. Each issue has its unique LD in the same way as the scenarios with NM tasks. The source code used to develop the scenarios is similar between both groups however, each cluster is evaluating different perspectives.

Two control questions were introduced inside the experiment, one to validate the interpretation of the source code and another regarding the general text reading. In embellishing the scenarios, the participants' level of experience was taken into consideration since not all had a very high knowledge profile in the Java programming language. Section 5.2 details the experiment conduction and how it interacted with each subject.

## 5.2 Experiment Process

For the conduction of the program comprehension experiment, a well-defined process was performed and divided into three main phases. Each step was designed to not interfere with subsequent steps. All the participants were told about the experiment process to ensure that they had acquired a basic familiarity with the experiment. A permission per subject had to be assigned before beginning to clarify the goals and risks of the experiment as detailed in the Annex A. Figure 31 presents the process elaborated to accomplish the controlled experiment and its objectives. Further, the principal phase and activities are defined as follows:

**Three phases** From end-to-end, the experiment process is broken down into three main phases. Phase 1 has the central point to extract characteristics via a questionnaire. It also details the information collected from the participant prior the experiment as well as for setting the EEG device. Phase 2 focuses on getting started in the experiment and explaining to the participant the process as well as collecting answers from each question in the experiment. Finally, Phase 3 has the objective to capture the participant's perception about the entire experiment process.

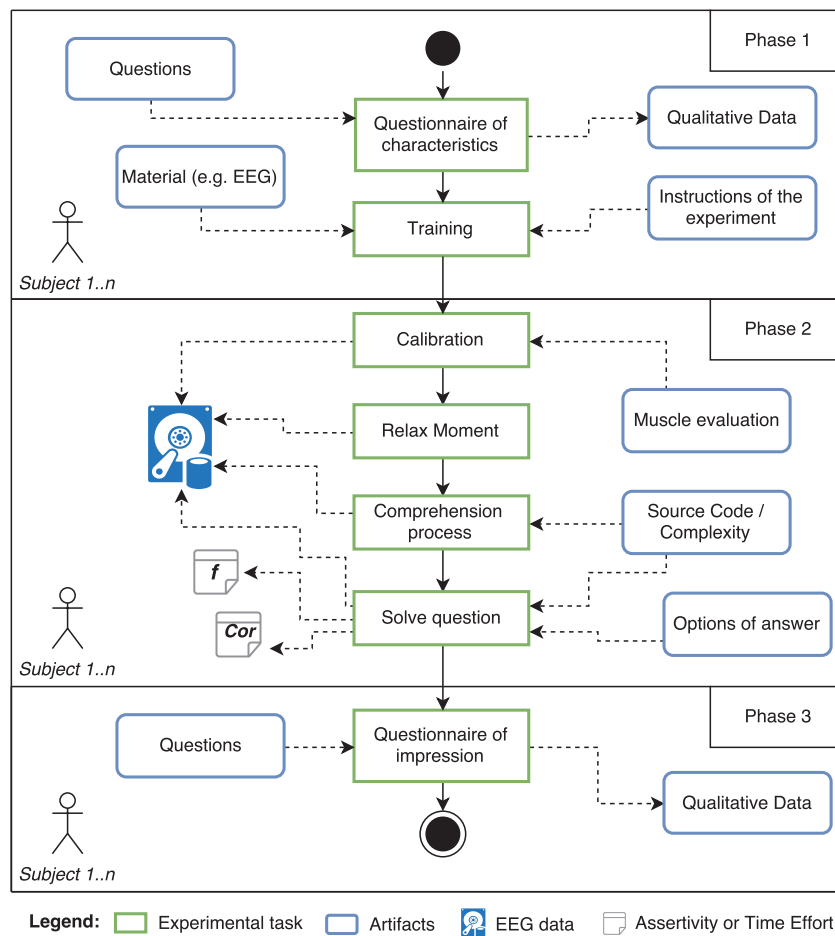
**Questionnaires** The participants received two questionnaires, one before the experiment and another after. The before questionnaire is provided in Appendix D, and it is called "Questionnaire of characteristics". The questions obtained data inherent to the participants' profiles, such as experience, academic formation, age, etc. The after questionnaire is provided in Appendix E and is called "Questionnaire of impressions". These questions focus on the analysis of the participant's perceptions regarding the experiment.

**Training** In the training task, the participants were instructed to ensure their awareness of the experimental process going through step-by-step. After answering the questionnaire, the EEG device was adjusted on the participants' head, and the behavioral instructions were given. A page of instructions was presented before the experiment explaining each relevant item and how each scenario should be handled.

**Calibration and relax moment** As the first step of phase 2, calibration plays an important part in understanding a participant's brain waves. The first part of the calibration process focuses on the eyes, and the participant is asked to open and close them for a duration of 30 seconds. The next part deals with the movement of the eyes around the screen and in each corner. After that, a relaxed moment occurs during 20 seconds was inserted prior each question to capture the current brain moment of the participant.

**Comprehension process and solve question** This experimental task appears after the relax time and it stays on the screen for a maximum of 60 seconds or until an answer is given. Five options for answers are displayed under each task, and the participant must select an option from 1 to 5 to solve the source code snippet. Each participant must resolve the task according to the source code presented in the question. The data collected is the resolution effort measured in seconds, the answer, the EEG data, and audios as well as screen video record. Additionally, participants individually complete every activity to avoid any threat to affect the results.

Figure 31 – Proposed experimental process.



Source: Created by the author

This section details the experimental process and material used in the controlled experiment. For that reason, subsection 5.2.1 introduces the participants, giving an overview of the characteristics and relevance in the IT area as well as their background in the university area. Subsection 5.2.2 presents the material used to execute the experiment and the hardware used such as a non-invasive electroencephalogram. Subsection 5.2.3 then highlights the orchestration of the execution of all details for the realization of the experiment.

### 5.2.1 Selection of Participants

This controlled experiment was carried out by 35 volunteers developers, who were selected by email recruitment and personal contact during the Master's program. In addition, the majority of participants are students of Applied Computing at Unisinos or connected in some way with the university. During the experimentation, the participants' characteristics, such as knowledge and experience, amongst others, were extracted with their permission. Regarding

experience in software development, there were both beginners and experts in the software development area, as well as from academia and industry. When analyzing the profiles of the subjects from the experiment, it was recognized that all of the individuals have a direct relationship with software development.

Table 15 shows the age range of the participants. Divided by group, there are two participants between 15 and 20 years, 13 between 21 and 25 years, 10 between 26 and 30 years, 2 between 31 and 35 years, 7 between 36 and 40 years, and 1 over 40 years of age. Table 15 presents that the 33 participants were male and 2 were female, which represents the current genders distinction. Another aspect detailed in Table 15 concerns the 12 participants who wear glasses, 3 who are left-handed, and the 2 ambidextrous subjects. These characteristics can influence how the human brain might behave even though this study did not notice it.

Table 15 – Age and characteristics of the subjects.

Range age	Quantity	Left-handed	Right-handed	Wear glasses	Male	Female
15 to 20 years	2	0	2	2	2	0
21 to 25 years	13	2	13	3	12	1
26 to 30 years	10	1	9	5	10	0
31 to 35 years	2	0	2	1	1	1
36 to 40 years	7	0	7	1	7	0
Over 41 years	1	0	1	0	1	0

Source: Created by the author

Table 16 details the participants' degree of education and the distribution among high-school, undergraduate, and graduate degrees. Over 98% of the participants presented in Table 16 are pursuing or have completed a higher degree of education. Besides, 57% of the participants are pursuing or have completed a master's degree or a Ph.D. degree. The participants' programs are computer science, information systems, analysis and software development, systems analysis, and digital gaming.

Table 16 – Participants' degree of schooling.

High-school		Undergraduate		Graduate (MBA/Master/Ph.D)	
Incomplete	Complete	Incomplete	Complete	Incomplete	Complete
1	0	10	4	19	1

Source: Created by the author

Among all the participants in the experiment, over 97.71% of them have at least two years of software development knowledge and a direct relationship with the technology area as shown in Table 17. From an academic perspective, 82.86% of the participants have studied for at least four years in university, and several did it in their areas of professional activity. At least 57.14% of the subjects have had low or higher experience in software modeling. Which displays that almost half do not use software modeling techniques in their daily professions.

Table 17 – Demographic data of participants (n = 35).

Characteristic	Answer	Quantity	Percentage
Software Development	<= 1 years	3	2.29%
	2 - 5 years	18	51.43%
	6 - 10 years	8	22.86%
	>= 11 years	6	17.14%
Current (Former) Position	Software development	16	45.71%
	Analyst/development	13	37.14%
	Other roles	6	17.14%
Software Modeling	Very low	15	42.86%
	Low	10	28.57%
	Medium	3	8.57%
	High	5	14.29%
	Very high	2	5.71%
Academic Study	<= 2 years	5	14.29%
	3 - 6 years	10	28.57%
	7 - 10 years	16	45.71%
	>= 11 years	4	11.43%

Source: Created by the author

Table 17 presents the current position of the individuals that participated in the controlled experiment. 83.85% of the current position of the participants are directly or indirectly related to program comprehension. Additionally, for the software modeling answers a Likert scale was used. The categories used are: very low (0-1 years), low (2-4 years), medium (5-7 years), high (8-10 years), and very high (over 10 years).

To better understand the participants, questions were added to extract the degree of knowledge using a Likert scale. In the Likert scale, the degree number one means very low, the degree number two means low, and it follows progressively until degree five, which is very high. The questionnaire is composed of 14 skills, involving: language, programming language, IT areas, and primary tasks such as system analysis.

Additionally, descriptive statistics were extracted from all skill types such as mean, SD, and median. In regards to the Java and the object-oriented skills, the means extracted was three, which displays that the audience has the knowledge to answer the scenarios. Four was the mean and median selected for programming logic and software development, proving that the participants had development skills.

Table 18 – Skill data of participants.

Skill Type	Mean	SD	Median	Likert scale quantity				
				1	2	3	4	5
JavaScript	2.60	1.22	2	7	11	10	3	4
R language	1.40	0.64	1	23	11	0	1	0
Python	1.65	0.82	1	18	13	2	2	0
SQL (ANSI)	3.25	1.09	3	2	6	13	9	5
C language	2.46	0.90	2	5	13	14	2	1
Object-oriented	3.23	1.29	3	5	5	8	11	6
JAVA language	2.80	1.14	3	4	12	9	7	3
UML diagram	2.70	0.88	3	1	16	11	6	1
Software development	3.86	0.83	4	0	3	6	19	7
System analysis	3.03	1.16	3	4	8	9	11	3
Back-end	2.97	1.30	3	6	7	9	8	5
Front-end	2.86	1.22	3	5	10	9	7	4
Programming logic	3.94	0.83	4	0	2	7	17	9
English	3.60	0.93	4	0	4	13	11	7

Source: Created by the author

### 5.2.2 Material

The list of materials usually required prior to the execution of a controlled experiment can sometimes be quite extensive, and the experiment executed in this study was no exception to this rule. This subsection describes the arrangements made before the controlled experiment's executions. A precaution taken was controlling the experiment environment around the participant for reasons that are going to be further described at the end of this chapter.

First, sounds and noises are known to distract a participant. Based on this, a known quiet room was selected, which means the sound did not cross 55 dB during the experiment. Internal light can affect and distract participants during an experiment. The light was controlled during all executions with it being on at all times during the day.

No electronic devices, including the participant's devices, were allowed within a two-meter radius around the experiment area. It is important to note that wave radiation can interfere with the EEG data, so electronics in the room were removed to create a high degree of isolation, including from the participant's objects.

Figure 32 presents the four components used in the controlled experiment, while a laptop model XPS 13 from DELL was used to collect the data and conduct the experiment. A wired keyboard in the QWERTY format was used to input the answer for each scenario as shown in Figure 32A. Specifically, only six keys were activated during the experiment: 1, 2, 3, 4, 5, and the space key. Also, the numeric buttons used in the keyboard were on top and not from the numeric keypad located at the side.

Figure 32 – Components for the experiment.



Source: Created by the author

Figure 32B details the EPOC+ neuroheadset, from the EMOTIV brand, used to collect EEG raw data from the subject using 14 channels at 256Hz and using three gyroscopes. This compo-



ment uses a wireless connection through technology Bluetooth version 4.0. Figure 32C presents a 19-inch monitor, model Dell E1909W, using the settings as brightness at 70% and contrast at 90%. The native resolution, which is 1440x900 pixels, was used in the experiment being refreshed at 75Hz. Figure 32D illustrates the EPOC+ receiver using a wireless connection to receive information directly from the headset into the laptop.

Among the items described above, other details should be highlighted as they also play a relevant part during the experiment. Both questionnaires were completed by using paper and a pen. This approach was used to keep the participant busy while the environment was adjusted for the experiment. Also, the entrance was blocked during the execution so no one could go in or out during the experiment to cause distractions.

The table used had the minimum items required to run the experiment since the participants could distract themselves unintentionally. An adjustable chair was used in order to make the participants as comfortable as possible. This point is crucial since muscular behavior impacts the motor area of the human brain, which would start working, generating unnecessary noise for further analysis.

### 5.2.3 Execution

During the experimentation process all the steps completed before, during, and after the experiments' executions were documented. The majority of the experiment executions were previously scheduled with the participant to obtain the most attention as well as minimally affect the psychological from the participant. For each new participant, a questionnaire of characteristics was provided. During this time, as the experiment was preparing for execution, which took approximately 7.96 minutes with a SD of 2.13 minutes. After, a consistent training was given taking approximately 4.5 minutes as well as the experiment itself which took 19.87 minutes and had a SD of 1.81 minutes. As presented in Table 19, the mean of the total elapse time used per each participant was 32.33 minutes with an SD 4.68 minutes.

Table 19 – Time consumed during the experiment.

Time (Minutes)	Questionnaires	Training	Experiment	Total
Mean	7.96	4.50	19.87	32.33
SD	2.13	0.75	1.81	4.68

Source: Created by the author

Figure 33 shows the experiment being executed by a participant that is a part of this research. The participant has agreed to be shown in this photo. This photo was taken after the subject's performance as this could interfere with the experiment. All participants repeated the



same arrangement, position, and process during the orchestration of the experiment. Each subject generated 10 trials during the experiment, which produced a total of 350 trials. A private laboratory was used, with space for one subject at a time, so no visual distractions occurred outside of the experiment.

Figure 33 – Participant conducting the experiment.



Source: Created by the author

### 5.3 Results

This section organizes and presents the results obtained during the controlled experiment. During the evaluation design, scenarios were chosen to explore different situations while understanding the source code. Also, EEG data was collected and applied to ARNI model to measure understanding using the *CognitiveEffort* technique. The calculation was done for the TE, the DR, and finally the CE metrics. The results achieved the objectives specified in Section 1.4 of this research. The results were input into the RStudio<sup>1</sup> tool, making the data available for the parameterization of the test with obtained results. The results accounted for clear evidence for the rejection of the null hypotheses as presented the subsections. Following the same point, the evidence points to the validity of the alternative hypotheses. The evaluations of all the questions were summarized in the Table 20, separated by the independent variable.

<sup>1</sup>RStudio: <https://www.rstudio.com/>

Table 20 – Descriptive statistic for the results (n = 35).

	Non-Modularity			Modularity		
	TE	CCR	CE	TE	CCR	CE
Min	11.080	0.000	-76.180	17.500	0.000	-189.489
25th	25.770	0.400	21.850	43.490	0.200	9.317
Med	36.960	0.600	40.130	59.780	0.300	34.265
75th	55.010	0.650	62.680	60.000	0.600	52.786
Max	60.000	1.000	83.410	60.000	0.800	82.874
Mean	38.840	0.531	37.920	51.910	0.350	26.575
SD	15.215	0.267	29.891	11.142	0.249	38.416

**Legend:** (*Min*) minimum, (*Med*) median, (*Max*) maximum (*SD*) standard deviation, (*TE*) temporal effort, (*CCR*) correct comprehension rate, (*CE*) cognitive effort, (*n*) participants

Source: Created by the author

The compiled results from Table 20 indicate that the non-modularity group had increased the CCR rate by 34.11% and reduced the TE by 33.65%, presented in Table 20. However, it was noticed that the CE had increased by 29.92% in the non-modularity group against the modularity group when all questions are summarized. Table 21 separates the information by scenario, displaying the mean result for each one and breaking down the information presented in Table 20. Based on the results collected, it can be observed that there was a significant difference, concerning the two code treatments in each task listed in Table 21. This affirmation is sustained by statistics in the following subsections.

Table 21 – Results per task and group using mean values.

Group Type	Variables	Task 1	Task 2	Task 3	Task 4	Task 5
Non-Modular	Temporal Effort	25.520	30.320	38.210	46.660	53.470
	Correct Comprehension Rate	0.688	0.719	0.531	0.500	0.219
	Cognitive Effort	34.240	43.350	43.890	35.600	32.530
Modular	Temporal Effort	51.150	41.070	56.980	52.500	57.830
	Correct Comprehension Rate	0.344	0.719	0.281	0.375	0.031
	Cognitive Effort	30.910	28.403	27.130	23.729	22.699

Source: Created by the author

Table 21 details each variable collected for the experiment, which was divided into two groups: non-modular and modular. In total 35 participants were part of the experiment. After an individual analysis, three of them were removed, since 45% of the samples had environmental noise. Three variables were characterized in the Table 21: temporal effort (TE), correct comprehension rate (CCR) and cognitive effort (CE). Looking at the results, it is perceptible that the TE increased as the tasks become more complex, from task 1 to task 5, but lower in all tasks in the non-modular group. Another point is related to DR, which was higher in almost all tasks across both groups, except in one case, task two, where it was the same.

Once the information was consolidated and organized the next step was to load the data into RStudio and determine the normality of the data in its distribution, as presented in Table 22. In this step, the Shapiro–Wilk test was used to validate if the data collected had normal distribution values (RAZALI; WAH et al., 2011). Table 22 details the result per question and presents values for  $p \leq 0,05$ . This result indicates that the data are not normally distributed, so a parametric test was executed, namely a student's t-test paired, including the Wilcoxon signed-rank test.

Table 22 – Results of normality test.

Task	Statistics	Shapiro–Wilk test		
		Temporal Effort	Correct Comprehension Rate	Cognitive Effort
All	<i>p-value</i>	<b>0.001</b>	<b>0.002</b>	<b>0.001</b>
	<i>W</i>	0.856	0.934	0.905
1	<i>p-value</i>	<b>0.001</b>	<b>0.001</b>	<b>0.034</b>
	<i>W</i>	0.858	0.636	0.959
2	<i>p-value</i>	<b>0.012</b>	<b>0.001</b>	<b>0.001</b>
	<i>W</i>	0.950	0.563	0.873
3	<i>p-value</i>	<b>0.001</b>	<b>0.001</b>	<b>0.001</b>
	<i>W</i>	0.828	0.624	0.918
4	<i>p-value</i>	<b>0.001</b>	<b>0.001</b>	<b>0.001</b>
	<i>W</i>	0.866	0.631	0.803
5	<i>p-value</i>	<b>0.001</b>	<b>0.001</b>	<b>0.049</b>
	<i>W</i>	0.567	0.388	0.964

**Legend:** (*W*) sum of signed ranks

Source: Created by the author

The student's t-test is considered a hypothesis test that uses statistical concepts to reject or accept a null hypothesis. It is used to compare means of different samples when following a normal distribution. Alternatively, the Wilcoxon test could be used as an alternative non-

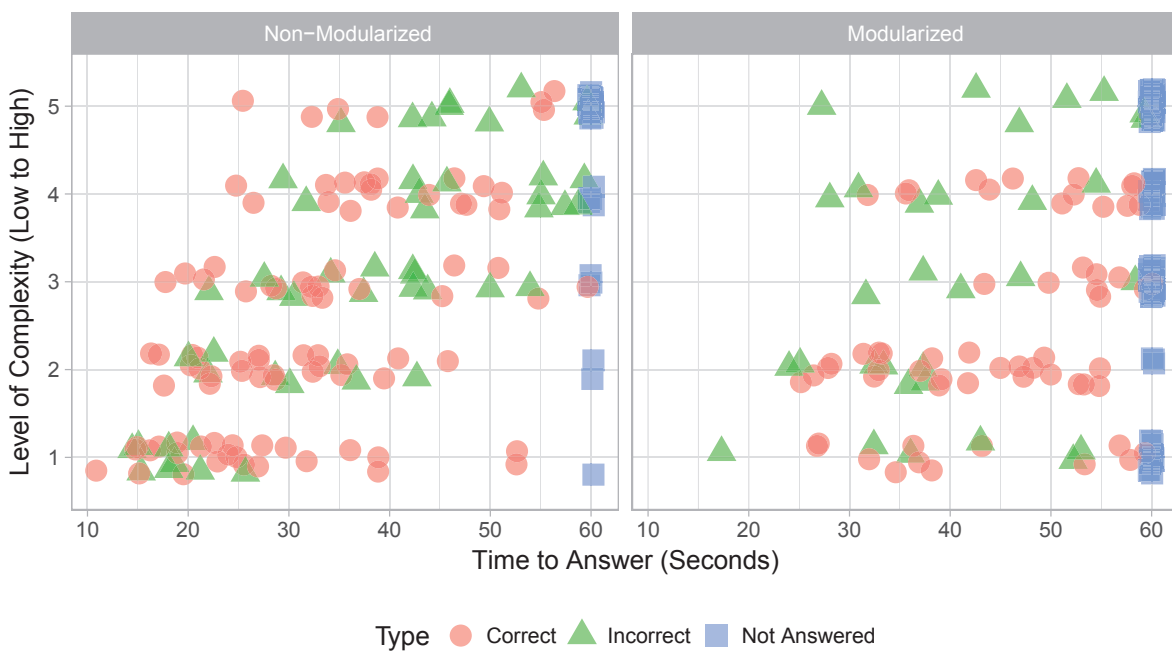
parametric static hypothesis test applied to match the means of two related samples such as paired difference tests. In this study, both statistical tests were used to test the hypothesis and provide statistical support for the results found (XIA et al., 2008).

This section is divided into three distinct areas: hypothesis 1: temporal effort, hypothesis 2: correctness, and hypothesis 3: cognitive effort. Each of these subsections will describe the results from different perspectives using the EEG dataset. First, the temporal effort will be made to see the brainwave activity and its behavior over time, based on the tasks answered. Next, correctness will be analyzed as a separate process to see the degrees and the results against the level of complexity. Finally, the cognitive effort will be validated using the ARNI model and the *CognitiveEffort* technique developed.

### 5.3.1 Hypothesis 1: Temporal Effort

The first hypothesis investigates the temporal effort made to comprehend the source code snippet. Table 20 details the descriptive statistics. The main finding indicate that the temporal effort from the participants to comprehend the answer was inferior in the non-modular group compared to the modular group, which is represented in Figure 34. Also, it graphically shows the trend per task complexity presented in the non-modularized code against the modularized group of tasks. Both groups present the same amount of points and they represent the answers.

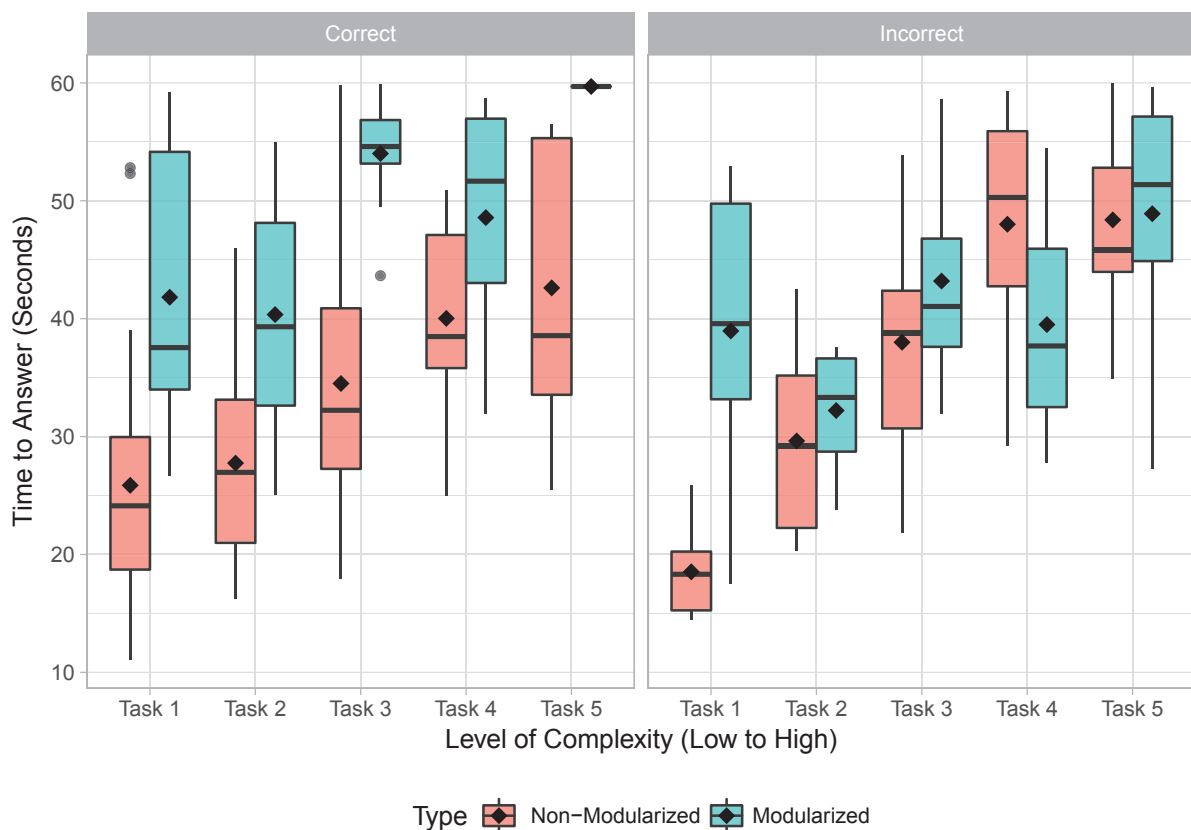
Figure 34 – Total temporal effort per task in the controlled experiment.



Source: Created by the author

To better understand the results, the Figure 35 shows answers divided in correct and incorrect answer type. In each task, the data are compared to answer type, making it easier to validate the data using a box plot graph type. In all of the correct answers, the temporal effort to solve the task is smaller in the non-modular code than in the modular code. For the opposite answer, the incorrect result details a huge difference in task 1, contrary to tasks 2, 3, and 5. Task 4 is the only question for which it took longer for a non-modular code to be understood against a modular source code. These results are an intriguing finding because common sense would be contrary. As an example, developers would spend less effort using a modular method to understand the source code.

Figure 35 – Answers and temporal effort per task.



Source: Created by the author

Table 23 presents the results obtained after performing the Wilcoxon signed-rank test and the paired student t-test. While the Wilcoxon test allows us to realize a pairwise comparison of the distributions, the paired student t-test was used to determine if two collections of data are significantly divergent from one another. It is crucial to perceive that the statistics were determined from a basis of 320 compositions. The significance of the results was at a level of 0.05 ( $p\text{-value} \leq 0.05$ ), as previously stated. The test using the mean variation among the

measures of the modular and non-modular groups, using various levels of complexity. Table 23 also shows the  $p$ -values for the pairwise correlation. Bold  $p$ -values symbolize statistically significant results and implies the rejection of the corresponding *null* hypothesis. This result shows that the subjects did not expend considerable temporal effort in understanding the source code in the non-modular instance. Therefore, the initial assertion that the modularity technique would not reduce the temporal effort at different levels of complexity was confirmed.

Table 23 – Temporal effort results of statistical tests.

Tasks	Wilcoxon test		Paired student t-test		
	$p$ -value	W	$p$ -value	DF	$t$ -value
All	<b>0.001</b>	9133	<b>0.001</b>	159	10.358
Task 1	<b>0.001</b>	504	<b>0.001</b>	31	7.763
Task 2	<b>0.001</b>	455	<b>0.001</b>	31	4.277
Task 3	<b>0.001</b>	460	<b>0.001</b>	31	8.369
Task 4	<b>0.003</b>	346	<b>0.001</b>	31	2.797
Task 5	<b>0.024</b>	144	<b>0.016</b>	31	2.242

**Legend:** (*W*) sum of signed ranks, (*DF*) degree of freedom

Source: Created by the author

Table 23 presents the results of the paired student t-test using the  $p$ -value ( $\leq 0.05$ ), which indicate that there is a significant difference between the mean ranks in repeated measures of temporal effort. For instance, in task 1, a  $t$ -value of 7.763 with  $p \leq 0.05$  indicates a statistically significant difference in the temporal effort associated with the non-modular group. Consequently, there is sufficient evidence to reject the *null* hypothesis and conclude that there is a discrepancy among the temporal efforts at the 0.05 level of significance. It is verified in this hypothesis that the temporal effort to read and understand the source code with the non-modular technique is smaller by 34.11% compared to the modular technique in all tasks.

**Conclusion of H1:** The main findings indicate that the temporal effort from the participants to comprehend and give an answer was inferior in the non-modular group when compared to the modular group in all tasks and levels of complexity.

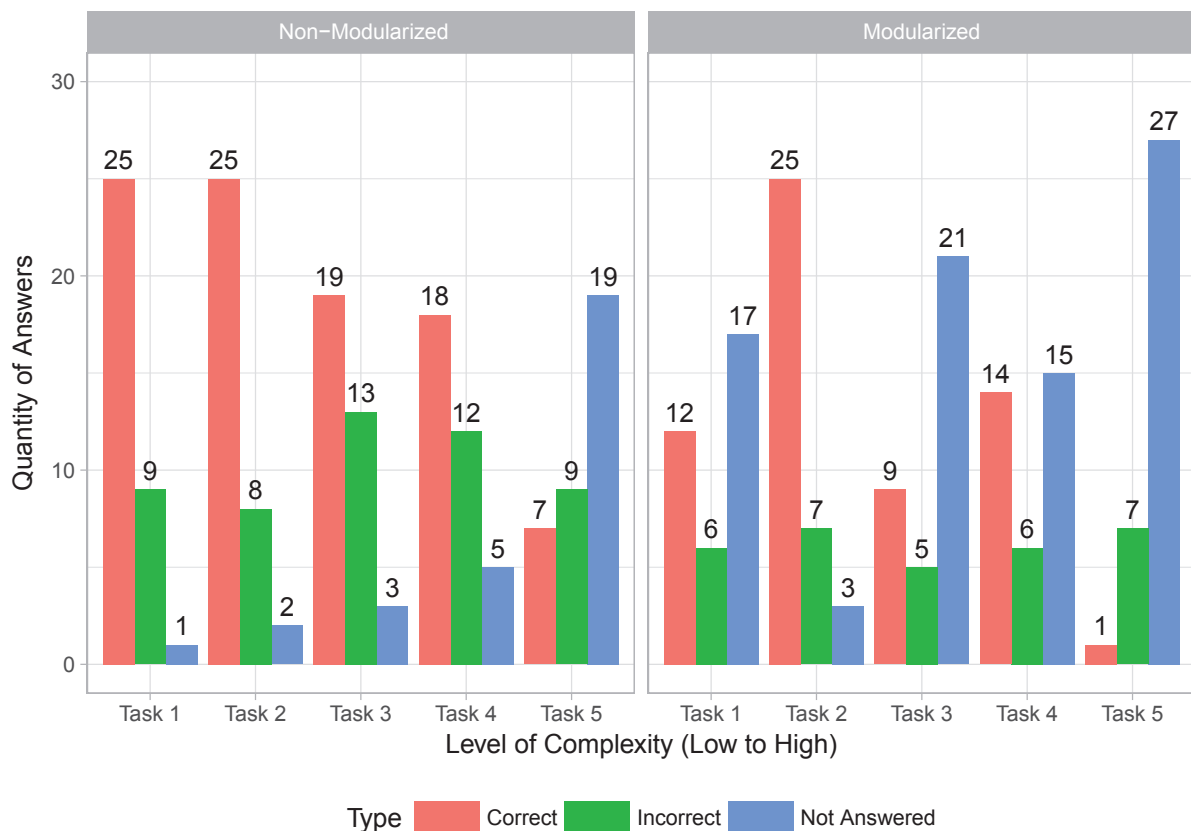
### 5.3.2 Hypothesis 2: Correctness

The H2 hypothesis analyzes the collected data concerning the impact of modularization on answer correctness. Table 20 details the descriptive statistics. Figure 36 presents the correctness



per task, with the y-axis representing the number of answers of a specific type. Also, the x-axis consists of the tasks applied in both groups of tasks. The histogram shows the correct modularity throughout the experimental tasks based on incremental degrees of complexity. A relevant insight was that the modularity techniques require different efforts at the lowest and highest levels of complexity, but at the same time, the non-modularity code does not guarantee superiority regarding the correctness in comparison with a balanced use of modularity.

Figure 36 – Responses to the questions during the experimental phase.



Source: Created by the author

Table 24 details the results where the modularization techniques assume a higher level of correctness when compared with non-modular source code. The Wilcoxon test was applied in the H2. The findings reveal the statistic and  $p$ -values for the pairwise comparisons. In the majority of circumstances, the  $p$ -value was smaller ( $p \leq 0.05$ ), so the *null* hypothesis of H2 can be rejected. The exceptions were tasks 2 and 4. The  $p$ -values of the five scenarios are shown in Table 24, with the  $p$ -values showing amounts lower than 0.05 in three tasks and higher in the other two tasks.

The results imply that there is a notable variance between the proportions of correctly understanding answers produced by the non-modularity code when compared with modularity

technique. It depicts the pairwise  $p$ -values for each measure. Bold  $p$ -values point out the statistically meaningful results, and it designates the rejection of the *null* hypothesis. It is appropriate to note that the sum of signed ranks ( $W$ ) determines the direction in which the result is significant. For instance, in task 5,  $W$  is positive 65, and the  $p$ -value is lower than 0.05 ( $p = 0.007$ ) for the measurement between both groups. This means that the inconsistency rate for the modular code is significantly lower when compared with the non-modular group.

Table 24 – Correct comprehension rate results of statistical tests.

Tasks	Wilcoxon test		Paired student t-test		
	$p$ -value	$W$	$p$ -value	DF	$t$ -value
All	<b>0.002</b>	310	<b>0.001</b>	31	3.931
Task 1	<b>0.002</b>	104	<b>0.001</b>	31	3.232
Task 2	0.500	28	0.500	31	0.000
Task 3	<b>0.010</b>	65	<b>0.007</b>	31	2.490
Task 4	0.143	68	0.146	31	1.072
Task 5	<b>0.007</b>	21	<b>0.006</b>	31	2.675

**Legend:** ( $W$ ) sum of signed ranks, ( $DF$ ) degree of freedom

Source: Created by the author

These results are further supported to apply the paired student t-test. All tasks obtained a  $t$  value of 3.931 with a  $p$  value lower than 0.001, which is lower than 0.05 and is thus a significant result. This result suggests that there exists a meaningful difference between the inconsistency rate using non-modular code. Nevertheless, acknowledging each experiment's scenarios, the results obtained did take valid significance ( $p \leq 0.05$ ). It is relevant to note that the degree of freedom ( $DF$ ) shows the direction in which the result is significant. It is also important to mention that all the results achieved were 33.65% better than the results in the modular code, using the mean values when compared with the modular group.

The results mean that the non-modular group significantly exceeded the modular technique in general. For example, in task 1 in Table 24, the  $W$  value of 104 with a  $p$ -value equal to 0.002 indicates that there is a significant difference in the non-modular group concerning the inconsistency rate. So, for the  $H_2$  of this experiment, there is statistical evidence to conclude that non-modularity technique impacts the effectiveness positively in several scenarios.

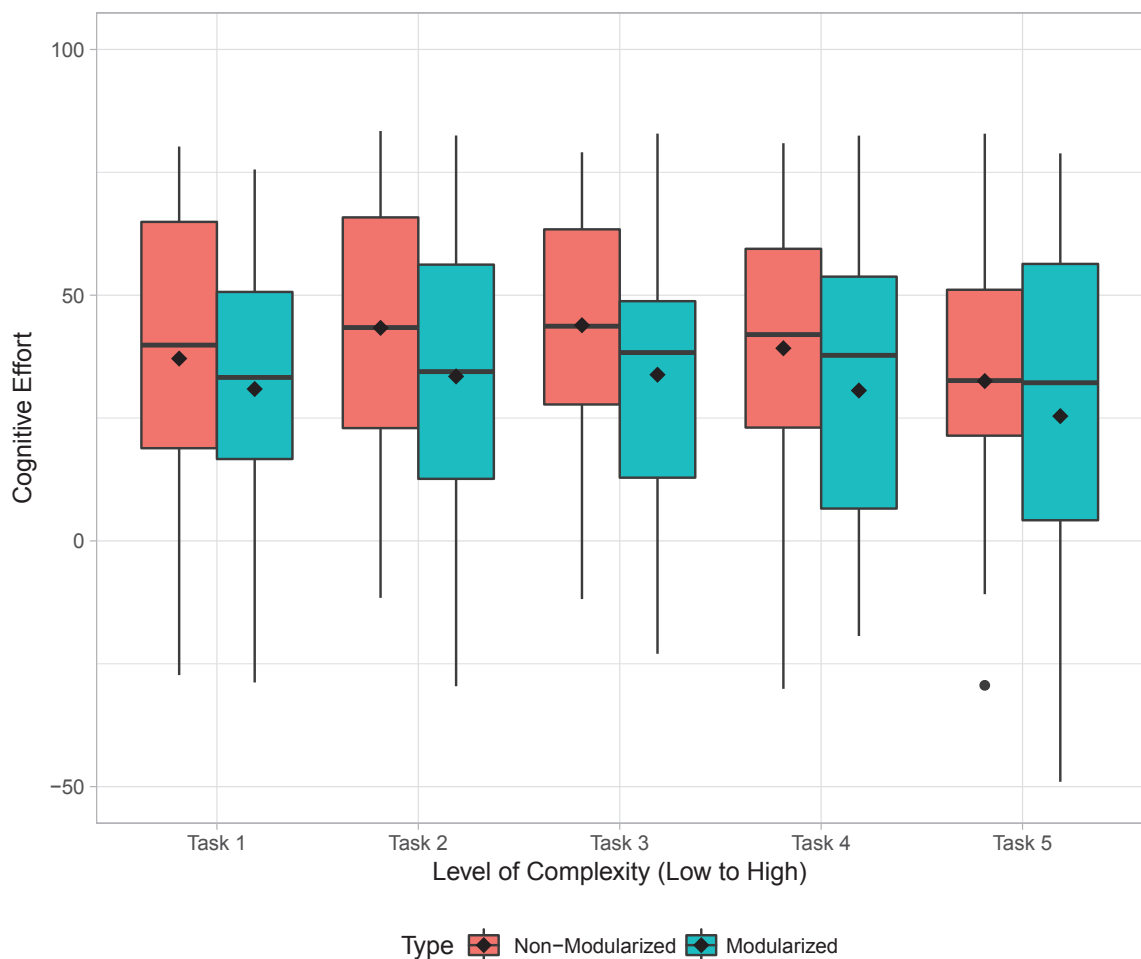
**Conclusion of H2:** The correctness rate for the non-modular group present higher values in general, but at the same time, it does not ensure superiority regarding the level of complexity in comparison with a balanced use of modularity.



### 5.3.3 Hypothesis 3: Cognitive Effort

The H3 hypothesis explains the results from the cognitive effort perspective and the influence in the developer's brain while it was analyzing the source code during the experiment. Table 20 details the descriptive statistics. It is expected to use the t-test paired since the same subjects were involved in both groups during the experiment, as well as in the Wilcoxon test. The principal finding is that the developers favor to invest greater cognitive effort to understand the source code using no modular technique rather than the non-modular, except for very low and very high levels of complexity (SCHAPIRO; HENRY, 2012; MACARTHUR et al., 2015). The result shows that they spent more cognitive effort for program comprehension in the non-modular code during the tasks 2-4. Figure 37 illustrates the cognitive effort comparisons generated by the ARNI model and using the *CognitiveEffort* technique. In addition, the difference is visually notable and statistically supported by two different statistical tests.

Figure 37 – Cognitive effort in the controlled experiment.



Source: Created by the author

The obtained data were evaluated against the paired student t-test and the Wilcoxon signed-rank test. Table 25 displays the details. The H3 uses the Wilcoxon test as well as the paired student t-test as statistical tests. The results expose the statistic and  $p$ -values for the pairwise comparisons. The level of 0.05 ( $p\text{-value} \leq 0.05$ ) was chosen as a significant value of the results obtained. In the general tasks, the  $W$  value obtained was 8675 with a  $p$  value below 0.001 and it represents a value lower than 0.05, which can be considered as being a significant result. The  $p$ -values are in bold font style, which symbolizes a statistically significant result, but at the same time, implies the rejection of the corresponding *null* hypothesis, with the exception of task 1 and task 5. Also, it shows from the test results that the subjects used more cognitive effort in understanding the source code in the non-modular group against the modular group.

Table 25 – Statistical results for Cognitive effort.

Tasks	Wilcoxon test		Paired student t-test		
	$p$ -value	$W$	$p$ -value	DF	$t$ -value
All	<b>0.001</b>	8675	<b>0.001</b>	159	3.881
Task 1	0.245	301	0.300	31	0.530
Task 2	<b>0.050</b>	350	<b>0.018</b>	31	2.196
Task 3	<b>0.005</b>	401	<b>0.007</b>	31	2.596
Task 4	<b>0.014</b>	382	<b>0.049</b>	31	1.606
Task 5	0.095	334	0.051	31	1.695

**Legend:** ( $W$ ) sum of signed ranks, ( $DF$ ) degree of freedom

Source: Created by the author

The paired student t-test was applied to further support the results obtained by Wilcoxon test. The general scenarios obtained a significant result, which is lower than 0.05, with a  $t$  value of 3.881 with a  $p$  value lower than 0.001. This result suggests that there exists a meaningful difference ( $p \leq 0.05$ ) around the median degree of complexity using modularity source code. It is also relevant to note that all the results achieved were 29.92% better in the modular code, using the mean values, when compared with the non-modular group. There is statistical evidence to conclude that the modularity technique affects the amount of cognitive effort positively in several scenarios for H3. However, the edge scenarios display a different behavior and point to there being no statistical significance amongst both groups.

**Conclusion of H3:** The major finding is that the developers favor applying a smaller cognitive effort to understand source code using the modular technique, except when exerting the smallest and largest degree of complexity.

## 5.4 Discussion

In this evaluation, the experiment seeks to explore different situations using modularity. This section provides highlights of the discussion based on the results and goals achieved from the controlled experiment. Data for qualitative analysis was collected from the following sources: questionnaires, screen records, audio, and personal perceptions. It is possible to obtain evidence to supplement the quantitative results and the outcomes from those pieces of information.

The collected data was grouped into two types of background experience, novice and expert, to better understand the background knowledge using the experimental data. Extra group division is explained in the following subsection. The *CognitiveEffort* technique uses weights to adapt itself to support different scenarios, such as program comprehension, but at the same time reduces or increases relevance in specific areas of the human brain. Concerning subject perception, a post-questionnaire was applied after the controlled experiment to collect information about the process and reduce threats connected with the way the experiment was designed. In conclusion, it can be concluded that the developed ARNI model, together with the results from the experiment, presents sufficient indication regarding the effectiveness of measuring the cognitive effort required for program comprehension using the *CognitiveEffort* technique.

In this way, with the help of statistical data and the results of the hypothesis analysis, a careful discussion is developed from different contexts. The remainder of this section is presented as follows: Subsection 5.4.1 compares the novice with the expert from a variables perspective. Subsequently, subsection 5.4.2 presents the different ways of collect and process human brain waves. Finally, subsection 5.4.3 presents participants' retrospective perceptions after the controlled experiment.

### 5.4.1 Novice and Experts

Background experience is known as one of the factors that can influence the results when the participants are confronted with one another. The goal of this subsection is to divide the participants involved in the experiment into two main groups: novice and expert. The experience level is essentially determined by the number of years that the participant has worked in software development. Based on that definition, the novice group is composed of 18 participants who have between zero and four years of expertise, while the expert group has 17 participants with five or more years of practice.

Table 26 details the data for each particular association as well as for each item of variable data collected in the experiment. The following list of measurements represent the descriptive statistic used to summarize the sample collected. It is also used to suggest indicators from different perspectives. The list of measures used are: minimum (Min), first quarter (25th), medium (Med), third quarter (75th), maximum (Max), mean, and the standard derivation (SD). The values in each variable respect the limits specified at the beginning of this chapter.

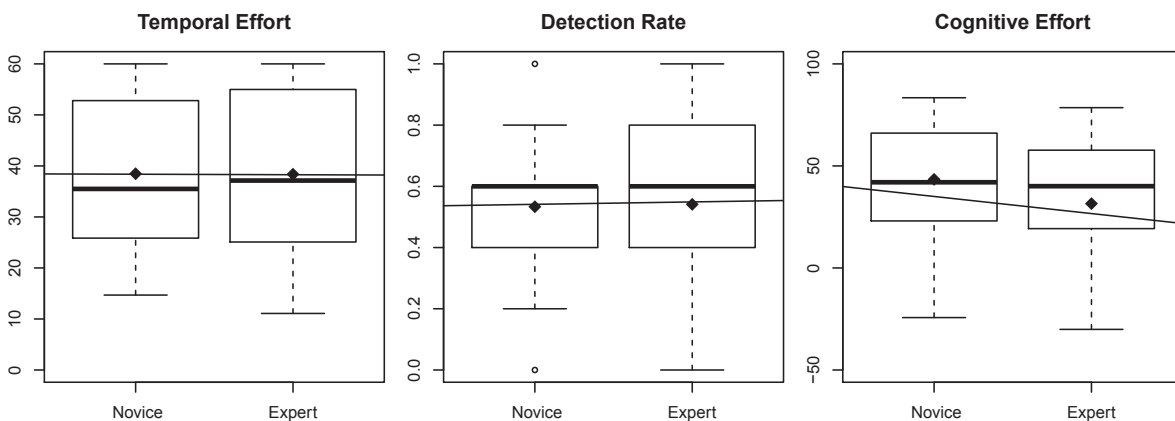
Table 26 – Participant’s comparison amongst novice and expert.

	Novice (n = 18)						Expert (n = 17)					
	Non-Modularity			Modularity			Non-Modularity			Modularity		
	TE	CCR	CE	TE	CCR	CE	TE	CCR	CE	TE	CCR	CE
Min	14.69	0.00	-24.32	25.17	0.00	-49.01	11.08	0.00	-30.10	17.50	0.00	-46.68
25th	25.89	0.40	23.05	39.14	0.20	7.51	25.09	0.40	19.27	47.04	0.20	12.70
Med	35.49	0.60	41.98	56.41	0.40	32.61	37.13	0.60	40.07	60.00	0.20	36.38
75th	52.68	0.60	66.07	60.00	0.60	55.95	54.99	0.80	57.71	60.00	0.40	48.40
Max	60.00	1.00	83.41	60.00	0.80	82.87	60.00	1.00	78.55	60.00	0.80	72.76
Mean	38.47	0.53	43.42	50.28	0.38	31.53	38.37	0.54	35.04	52.30	0.32	36.38
SD	14.91	0.25	26.39	11.47	0.24	31.78	15.36	0.28	28.45	11.55	0.25	26.90

Source: Created by the author

Figure 38 illustrates the data from Table 26 from the non-modular group for each variable comparing the novice group to the expert group. The temporal effort calculates the total time needed to comprehend the source code fully. The results for the novice group are almost the same as those for the expert group. From the perspective of the correct comprehension rate, the means of both groups are almost the same, except they differed in boundaries results. When comparing the values for cognitive effort, it is possible to notice that the amount of brain activity is reduced, which most probably could be attributed to the experience spent in development.

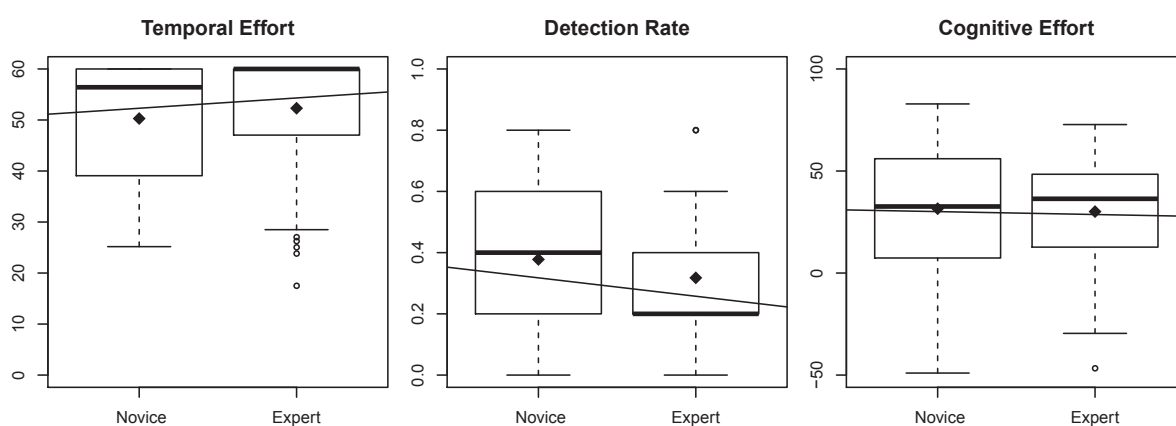
Figure 38 – Non-modular Results grouped by experience



Source: Created by the author

Figure 39 illustrates the modular group for each variable, comparing the novice and the expert groups by applying the data from Table 26. The values of TE obtained from the modular group show that the expert, on average, took longer to answer the scenarios than the novice. From the CCR viewpoint, the expert had a higher rate than the novice, not only on average, which could represent that experts are more familiar with this approach. Finally, the CE for both cases demonstrated similar values, except that the novice had the highest and lowest quantity of cognitive synapses throughout the experiment.

Figure 39 – Modular Results grouped by experience



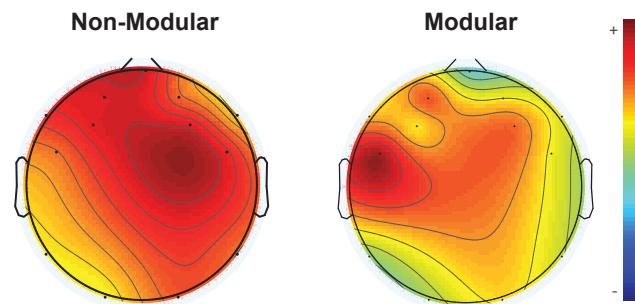
Source: Created by the author

Based on the analysis of the results, it is reasonable to suggest that the experts better handled the non-modular code, and the novice participants better handled the modular code. This can be explained by the fact that the novice starts from the beginning with the concept of modularity and breaks down large pieces of source code. However, more studies should be executed to understand the participants' behaviour by using senior and novice groups with modularity techniques. Nevertheless, it becomes evident that the modularization source code has some impact, positively or negatively, depending on the context applied.

#### 5.4.2 Brain Activity

Dozens of thought synapses can occur every second in a human's cerebrum. Each synapse reproduces communication amongst each neuron to formulate a thought in our brain. These subsections focus on brain activity and its behavior during the program comprehension. Figure 40 illustrates the brain's activity for the same participant in each group studied, both non-modular and modular. The most obvious impression is that the non-modular group shows that the color red has a much stronger presence than the modular code. The highest position indicates that modular used the working brain area and non-modular the concentration and planning area.

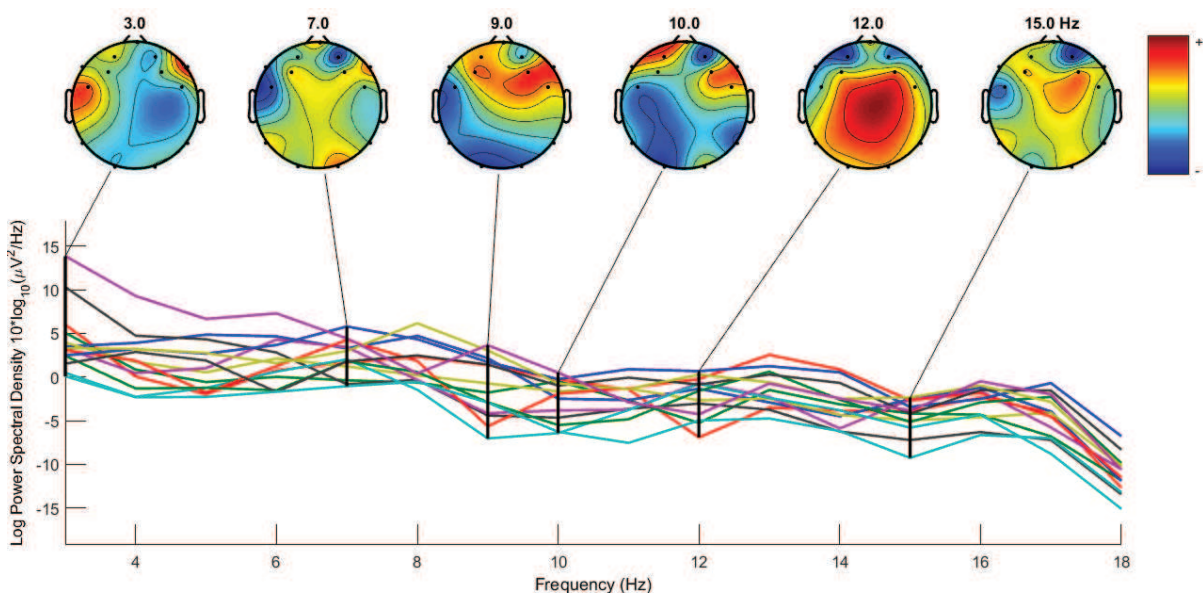
Figure 40 – Scalp comparison using group’s data from the experiment.



Source: Created by the author

EEG technology has the potential to capture very deep waves generated by groups of neurons in different areas of our brains. So, it is essential for any technique that intends to use EEG to isolate and process each sensor individually. Figure 41 shows, for the 12 Hz non-modular scenario of a participant, the frequency behavior used to generate Figure 40. Also, the range of 3 Hz to 18 Hz was selected because it has been considered the most relevant for extracting cognitive thought from humans and has shown a direct relationship to mental tasks. This figure shows that it was the most relevant frequencies in this experiment. The PSD details the power for each of the frequencies presented on the scalp: 3, 7, 9, 10, 12, and 15 Hz. The blue and red color means low and high degree of brain activity respectively.

Figure 41 – Frequency behavior in the non-modular group during a scenario.



Source: Created by the author

The ability to adapt itself to a different context can be considered a major property, and the developed ARNI is capable of accomplishing this, through the *CognitiveEffort* technique. The results above confirm that the results found in the hypothesis section reaffirm the quality and precision of the data in the experiment. The rest period, represented by the baseline, basically eliminates the majority of the psychology and physical behavior, and it allows much cleaner data to be collected from the human brain. It is relevant to mention that the rest period plays a major role in the ARNI model developed in this research. Lastly, this study developed a new way to measure the developer's thoughts during program comprehension.

### 5.4.3 Experiment Retrospective

After the experiment, the subjects answered a post experiment qualitative questionnaire, which served to collect their perceptions about eight questions. Table 27 details the questions asked in the questionnaire. Each question addresses key issues controlled during the experiment, including time to answer the questions, level of difficulty, among other. More details about this questionnaire can be found in Appendix E.

Table 27 – Questions in the qualitative questionnaire.

Question	Description	Answers
Q1	The time was enough to execute the questions of the experiment.	B
Q2	The task of each question was perfectly clear to me.	B
Q3	The questions answered in the tasks were perfectly clear.	B
Q4	Judge the difficulty of the tasks related to understanding the code in JAVA language.	A
Q5	Judge the difficulty of pressure-related tasks to respond.	A
Q6	The font and style of the text was clear for understanding the source code.	B
Q7	The task options made it difficult to guess the correct answer without fully understanding of the source code.	B
Q8	The instructions given, prior to the experiment, were clear and understandable.	B

**Legend:** (A) very low, low, medium, high, very high  
(B) strongly disagree, partially disagree, indifferent, partially agree, strongly agree

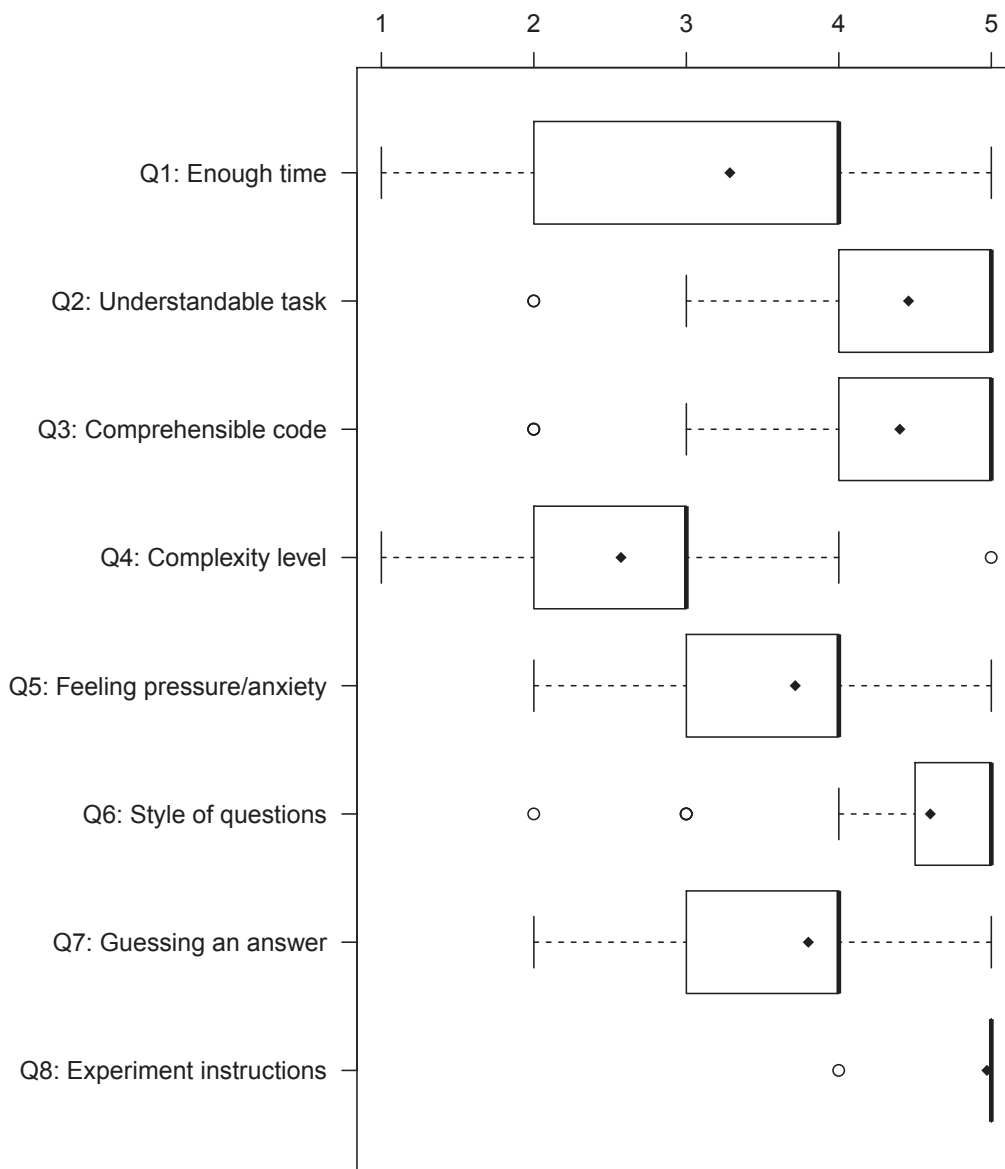
Source: Created by the author

Figure 42 shows the high-level subject's perceptions from the experiment using the Likert scale (1-5). Each question had answers from very low to very high (1-5) or strongly disagree to strongly agree (1-5). The results reveal the high level of pressure (Q5) during the experiment for

the participants. The subjects think (Q1) the elapsed time fluctuated based on the background knowledge of the participant. The data shows that the participants understood the processes during the experiment in time (Q2), explanation (Q3), style (Q6) and instruction (Q8).

The complexity (Q4) in the majority was low or regular, which should ensure that they were not the focus but the developer’s understanding. It is important to reinforce that the participant was not asked for additional information relating to their answers. It was difficult for the high majority who answered the scenarios to guess (Q7) a reply since the experiment was designed to make it difficult to guess based on the options. Furthermore, there is sufficient evidence to guarantee the quality of the controlled experiment from the participants’ perspective.

Figure 42 – Post experiment qualitative questionnaire using Likert scale.





## 5.5 Threats to Validity

As in any study involving experiments, threats to validity may potentially impact the process used and the results achieved. This study is not an exception to that rule. This section discusses the approaches used for handling threats and focuses on mitigating issues that might affect the results achieved (FARIAS; GARCIA; LUCENA, 2014; FARIAS et al., 2015b). It examines possible threats from the evaluation perspective and addresses solutions to mitigate it.

Each risk was mitigated using the most appropriate mitigation technique in order to reduce the potential threat in this study. It cannot be concluded that the results found can be generalized to any degree of source code since those selected for this experiment are considered small (BIFFI; TUISSI, 2017). Also, it cannot be ensured that the results are equivalent to the more explicit manipulation of independent variables since several variables are involved in the experiment, such as subject randomization as one example.

This section is divided into four validity subsections. First, Subsection 5.5.1 discusses the conclusions in this study and details with what it achieves and compares them with its goals. Second, statistical validity concerns related to statistical models are discussed in detail Subsection 5.5.2. Third, the internal validity of the conclusions derived from the data are covered in Subsection 5.5.3. Lastly, the external validity handles the conditions that support the generalization of the results, which is covered in Subsection 5.5.4.

### 5.5.1 Construct

The construct validity concerns the degree of inferences from the observation of stimulation and outcome incorporated in this study (JORGENSEN, 2005). Particularly, researchers evaluated (1) the experimental area that collected the EEG data, (2) whether the quantification methods of the dependent variables were appropriate, and (3) whether the process and the accuracy were precise.

First, the characteristics of the program comprehension did not require any specific knowledge regarding development tools. With this in mind, it was observed that simple text composition helped to minimize problems related to the software development tools. Nevertheless, by selecting only one programming language, in our case JAVA, there is a potential increase of risk as people think differently depending on the language. This risk should be noted. Despite the complexity of the experiment, the scenarios were kept simple and as clean as possible.

Another risk that was managed was related to the composition of the scenarios, which may unintentionally have influenced the results. The font or pattern used in the questions did not impose threats to the validity of the empirical results. The researchers made decisions about the experiment and sought a study design that reduced external influences on the results. Finally, it was observed that the pattern of the results did not agree with the empirical findings, as well as the number of scenarios developed in each group.

The EEG technology brings with it several threads, in particular, with regards to the environmental settings. The electrical waves may interfere with those generated from the brain, when the experimental environment is not well isolated, and are manual, similar to that of electronic devices. The guidelines were checked, to better identify and avoid any external conflict. The wave threads were completed and mitigated in a laboratory with a two-meter radius. They were isolated, and no electronic equipment was allowed within the radius. The lights and the sound in the room were controlled, so as not to generate any distractions during the experiment. These precautions were reviewed during the pilot experimental test, in order to make sure that the environment was isolated, and that it would not interfere with the collected data.

### 5.5.2 Statistical

This subsection details the significant concerns about the statistical tests and their mitigation. In addition, the covary strength, as well as the cause and effect covary, presumably, were validated (HYMAN, 1982). In terms of the initial judgments, they might be incorrectly defined, such as the importance of covariation and the level of reliance that the evaluation permits. Additionally, the presence of the second inference might improperly assume a causal relationship linking the selected variables when, in reality, they do not exist (CAMPBELL; RUSSO, 1999). To minimize the possible threats to the results, it was verifying proper statistical approaches based on independent and dependent variables selected.

The risks were reduced to the causal relationship among the variables examining the normal distribution of the obtained trials. For this purpose, it was reasonable to test if the data should be used by parametric or nonparametric statistical methods. Thus, the Shapiro-Wilk test was adopted to verify the normal distribution of the data captured (PETRUCCELLI; NANDRAM; CHEN, 1999). Concerning the statistical test applied, the results present positive outcomes that provide enough information so as they were applied correctly, and it is presupposed the tests were not broken in any way.

The controlled experiment tried to obtain the maximum number of samples and consequently improve statistical efficacy. The significance level at 0.05 level ( $p \leq 0.05$ ) was admitted to test every hypothesis from the statistical aspect. It followed well-known guidelines to enhance conclusion validity. The designed scenarios attempt to represent genuine pieces of source code applying different degrees of modularization. Various improvements were made to overcome probable errors that could wrap the causal amongst the variable under investigation (TROCHIM; DONNELLY, 2001).

### 5.5.3 Internal

This subsection will discuss the major details about the analysis performed on the results. The variety of distinct elements may directly impact the performance of the individual and influ-

ence additional circumstances (WOHLIN et al., 2012). Validity reflects the potential to obtain the accurate judgment based on the results of the controlled experiment. The questionnaires data transferred to the statistical system was double verified and the EEG data was completed automated to avoid any threats. The EEG device was tested and executed to obtain assurances that the data is accurate, as well as environmental settings were reviewed.

The cognitive activity may seem as the major threat to the internal validity. The independent variable is unknown, and it unexpectedly affects the dependent variable. Thus, a pilot study was carried out to learn and isolate the baseline at the current moment to ensure that the dependent variables were affected as least as possible since it is impracticable to isolate every external variable (SHADISH; COOK; CAMPBELL, 2002).

Pilot tests were executed to confirm that the independent variable exclusively influences the dependent variables. It was perceived that the obtained results for the assertiveness and temporal effort were confidently generated by the scenario developed of the source code used as well as the cognitive effort calculated. Despite this, a few threats were recognized through this process. In addition, an ever-present concern throughout the study was to make sure that the EEG data collected in this study was clear and represent subjects cognitive and can be analyzed by multiple participants. Afterwards, the risk was mitigated by establishing a measurement baseline among participants and two-rounds of data was done to complete the review.

#### 5.5.4 External

This area concerns the external validity transfer and the validity of the collected results in other widespread contexts (MITCHELL; JOLLEY, 2012). The material and devices utilized in the controlled experiment are equivalent to the one that was used in the industry field. In addition, the generalization of the results from the experiment was acknowledged by academic and industrial perspectives. This adds value to realistic domains.

There is an extent to which the results of this study should be generalized to other realities, such as using various levels of modularity, using more experienced developers, and quantifying other inconsistencies. Thus, the data analyzed has examined causal relationships and its fluctuations in customs, personalities, and other contexts. As this study has not been replicated yet, it is recommended that it be replicated in order to confirm its generalization for program comprehension context as well as similar studies.

It is necessary to highlight how the scenarios developed and employed in the experiment were minor compared to the ones used in industry. In addition, to mitigate the movement threat partially, it used an EEG device that supports wireless connections and allows the developer to move freely as he usually would do. Given the criteria listed in this study for software development, it is safe to conclude that the results achieved may be generalized at some point in very similar contexts. Nevertheless, several limitations should be taken into account, and additional experiments should be done to ensure a safe level of generalization.



## 6 CONCLUSION

This study addressed the problems and limitations of using BCI devices to measure program comprehension. Also, this process detects brain activity while software engineers read the source code. The SMS has been adopted to highlight the gaps regarding the lack of a panoramic view as well as to comprehend the existing literature. The SMS retrieved a total of 1709 articles, which were thoroughly refined into 12 articles and labeled as primary studies. The primary studies were the cornerstone for the formulation of the RQs defined in Section 1.3. The main objective of this study was to investigate and propose a model to measure program comprehension. Additionally, future challenges are pointed out, as well as final considerations and limitations of the study.

After obtaining an overview of the published articles, the next step was to dive into the main theories to acquire a better understanding of the developer's brain. The background explored the program comprehension concepts, such as mental models and factors that influence understanding. The psychophysiology explored the brain and body relation using biometric devices such as BCI. An non-invasive EEG device was then carefully detailed with wave formats, and the type analysis was applied. Implementing the ARNI model was the next step performed. In this step, the architecture was designed with the support of EEG devices in mind. FFT and ERD methods were used to process and classify the brain wave patterns generated by the programmer as well as the *CognitiveEffort* technique.

The experiment was conducted involving 35 software developers to understand ten distinct source codes snippets. The questions were divided in two main groups: module and non-module. Each group had five levels of complexity. Two pilot questions were inserted before the experiment to ensure the participant understood the process. Two control questions evaluated if the subjects were attempting to understand. The EEG data from the experiment, as well as the timing and assertiveness, were then captured. The ARNI model analyzed the data and then produces metrics of program comprehension. The main impact of this study opens a new frontier of measuring the impact of the source code from an alternative perspective.

This chapter is organized into three sections as follows. Section 6.1 presents a summary of the contributions obtained in this based on the defined RQs. Section 6.2 highlights the limitations and precautions taken to handle those through this study. Finally, Section 6.3 shows the opportunities for future work after the conclusion of this research.

### 6.1 Contributions

This section presents the contributions acquired during this study according to the RQs specified in Section 1.3. On a high-level perspective, there are three main contributing factors. First, this study executed an SMS in the program comprehension field applied to BCI devices. Second, the ARNI model was developed to measure a developers' comprehension of source codes

using the CognitiveEffort technique. Lastly, a controlled experiment was executed to validate the ARNI model and generated empirical results. Each item below describes the process used to enrich the program comprehension field and to improve what is known about the software developer's brain.

**The state-of-the-art regarding program comprehension in software development using BCI devices (RQ-1)** An SMS established the well-defined protocol for the search for studies related to software engineering and neuroscience. The SMS created a panoramic view of program comprehension and BCI devices in literature. Empirical knowledge was created by organizing the SMS data concerning respective specialties. The studies were categorized regarding the place of publication, the level of maturity and the methodology employed, and gaps were then identified. The SMS results brought a direction for future challenges in the area of program comprehension models using EEG devices. Additionally, the result exhibit the areas searched and fields including the potential for exploration based on the domains investigated.

**Estimation of the cognitive effort collected by BCI devices to measure the understanding of source code in developers (RQ-2)** After confirming that there is a gap in measuring program comprehension in literature, the ARNI model was proposed to address this gap. The ARNI architecture was designed to be flexible and support different EEG devices and various software languages. The ARNI model is based on three individual components to measure program comprehension: EEG markers, pre-processing and wave pattern analysis. The architecture definition is also a process to work with a software developers brain in multiple contexts. The ARNI model has a technique to translate the brainwave to follow three criteria, FFT, ERD and CognitiveEffort technique. The theta and alpha wave groups translate the importance in the wave patterns. Furthermore, the *CognitiveEffort* manipulated the weight for each EEG channel selected to extract the brain's effort during the comprehension.

**The impact of modularization on the cognitive effort (RQ-3)** The cognitive effort is defined in the literature as the amount of brain energy needed to accomplish a thought. Modularization is formalized through a degree that components may be separated, making the source code clear and precise. Hence, a controlled experiment was developed to estimate the cognitive effort, from the developers' context, which was the understanding of source code in distinctive degrees of modularity. The main finding reflect that the timing and assertiveness are higher for non-modularity programs than the modular ones. The results for cognitive effort concludes that code with extreme levels of complexity can result in higher difficulty for completion in modular code compared with non-modular. Also, the stability of non-modularity and modularity code is the core for solid and securer comprehension of source code.

## 6.2 Limitations of the Study

In studies that concern the human brain, it becomes imperative to acknowledge that limitations exist during the process, in particular with the experiment and development of new

models. Since the development time for this research was limited, different approaches were used to try to cover the majority of any possible gaps. This study covered, in its course, essential questions regarding cognitive effort in a software developer's brain and its effect based on the understanding of the source code. Thus, the central weaknesses in this study were then classified. They are itemized below with their mitigation.

**Electroencephalography** The EEG devices may suffer from various stimulus when it is worn on a human scalp. Threats such as noise from muscles, the environment, and because it is a low-cost device can be applied in this study. Focus on mitigating the noise from muscles and the environment; a restricted room and protocol were created to minimize these effects. Also, the best low-cost device was used since a wireless device, and the financial points were crucial from the industry's sector.

**ARNI Model** The ARNI model developed in this study center on the known development environment and the use of a wireless, non-invasive EEG device. Although it was developed thinking in the most generic cases, some potential threats and limitations can be applied. Focus on mitigating the generic model, as the algorithm broke down in three main parts which made it easier to be manipulated in an atypical scenario.

**Cognitive Effort Technique** The technique developed to measure program comprehension prioritizes some channels over other ones using weights. Potential threats may be generated when values are calculated using the same channel. To mitigate the case one channel being used, the four main channels in the cognitive area were used based on several experiments. The channels that were emphasized are areas with less noise and higher data quality to calculate cognitive effort.

**Controlled Experiment** Several variables can influence any experiment involving EEG devices. First, there is a need to expand and increase the number of participants related to the industrial sector in particular. 35 participants were involved in the controlled experiment to mitigate part of that issue as well as the replication of this experiment for further validation. Another point involves the results collected and generated by the ARNI and the cognitive effort technique. Focus on mitigating this issue, and statistical tests, such as the Shapiro-Wilk test, Wilcoxon test, Paired T of student test, and Cohen's d effect were executed in all the data to add extra validity and enhance the comprehensiveness of the data produced.

### 6.3 Future Works

Notwithstanding the improvements and enrichment provided by this study, it is expected to pave the way for a more ambitious agenda regarding the challenges to better comprehend brain activity. This section intends to extend ideas for future works that have arisen during this study on the neuroscience area applied to the computer science field. Several major challenges in the field of program comprehension deal with brain activity. A few research areas, itemized below, focus on the developers' brain for continued exploration in this field.



**Cognitive dimensions** Abstraction is known as the basis of cognitive dimensions, and it can be described as one of the principles for the user interface (UI) in programming language design. Based on other existing cognitive dimensions, such as consistency and diffuseness, which dimension have the potential to improve the comprehension in a language development as well as reducing the overload in the working memory phase.

**Brains perception with MDD** Model-driven development (MDD) has been changing the way source code is automatically generated in the industry. In the last decade, various studies have pointed to a software development environment using only visual components such as UML diagrams. In this context, a new field role of visual items opens up for exploration. How the developer's brain behaves when understanding source code against diagrams and how large is the overload thinking about the code that will be generated.

**Understanding syntax** In software development, a problem can have multiple solutions that can be developed in many different ways. Numerous development languages have more than one way of writing the same functionality using different syntax. It becomes needed to know which syntax might require more brain processing to be executed than others for the same task. What is the difference when a feature is converted from one syntax format versus another, and how might syntax affect the cognitive memory in the program context.

**Privacy and ethics concerns** The BCI technology has to become more accessible than ever for final users. The approach, such as the recruiting process, as one example, which measures expertise or the level of a specific skill. Since a human has limited control over how the brain works, what are the limitations from a company's perspective, and also, which methods can ensure the privacy of thoughts in the digital world and their degree of security.

**Educational context** In academia, there are several techniques to teach how logic works and how to develop software and writing source code, but no agreement exists about what the best methodologies are. Different methods used to teach software language should be identified and classified as the student's brain uses BCI devices and even biometric devices.

**Big Data and IoT** EEG devices are known for generating huge amounts of information. Just a few devices in a median sample frequency can fill regular hard drives in a matter of hours. Hence, it is difficult to determine what the best techniques are for manipulating huge amounts of EEG data in the internet of things (IoT) environment. Also, pattern analysis systems might look for similarities or patterns using large data sets produced.

**Overcoming developers capacity** In their everyday work as a programmer, there is a point when a developer's brain crosses its limit after too much thinking. In this sense, what would happen if we could have a machine process our thoughts instead of a developer's brain? Scenarios when a developer has to keep in mind diverse software contexts and variable values while debugging may be supported by a computer. Devices such as BCI, and special EEG ones, could enhance the communication between a human and a computer.



## REFERENCES

- ALLANSON, J.; FAIRCLOUGH, S. H. A research agenda for physiological computing. **Interacting with Computers**, Oxford University Press, v. 16, n. 5, p. 857–878, 2004. ISSN 09535438. Cited on page 44.
- ALTMANNINGER, K.; SEIDL, M.; WIMMER, M. A survey on model versioning approaches. **International Journal of Web Information Systems**, Emerald Group Publishing Limited, v. 5, n. 3, p. 271–304, 2009. ISSN 1744-0084. Cited on page 76.
- ANDERSON, E. W. et al. A user study of visualization effectiveness using EEG and cognitive load. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [S.l.], 2011. v. 30, n. 3, p. 791–800. ISBN 1467-8659. ISSN 01677055. Cited 2 times in the page 52 and 53.
- ANDREASSI, J. L. **Psychophysiology: human behavior and physiological response**. [S.l.]: Psychology Press, 2000. 538 p. ISBN 0805849513. Cited on page 44.
- ARISHOLM, E. et al. The impact of UML documentation on software maintenance: An experimental evaluation. **IEEE Transactions on Software Engineering**, IEEE, v. 32, n. 6, p. 365–381, 2006. ISSN 00985589. Cited on page 31.
- ARISHOLM, E. et al. Evaluating pair programming with respect to system complexity and programmer expertise. **IEEE Transactions on Software Engineering**, IEEE, v. 33, n. 2, p. 65–86, 2007. ISSN 00985589. Cited on page 52.
- AYRES, P. L. Systematic Mathematical Errors and Cognitive Load. **Contemporary educational psychology**, Elsevier, v. 26, n. 2, p. 227–248, 2001. ISSN 0361-476X. Cited on page 45.
- BADDELEY. Is working memory still working? **American Psychologist**, American Psychological Association, v. 56, n. 11, p. 851–864, 2001. ISSN 0003-066X. Cited on page 52.
- BADDELEY, A. Working memory: The interface between memory and cognition BT - Memory Systems 1994. **Memory Systems 1994**, MIT Press, v. 4, n. 3, p. 351–367, 1994. Cited on page 52.
- BAHLMANN, J.; SCHUBOTZ, R. I.; FRIEDERICI, A. D. Hierarchical artificial grammar processing engages Broca's area. **NeuroImage**, Elsevier, v. 42, n. 2, p. 525–534, 2008. ISSN 10538119. Cited on page 48.
- BALDUCCI, F.; GRANA, C.; CUCCHIARA, R. Affective level design for a role-playing videogame evaluated by a brain-computer interface and machine learning methods. **Visual Computer**, Springer, v. 33, n. 4, p. 413–427, 2017. ISSN 01782789. Cited on page 44.
- BAŞAR, E. **Brain Function and Oscillations: Volume II: Integrative Brain Function. Neurophysiology and Cognitive Processes**. [S.l.]: Springer Science & Business Media, 2012. Cited on page 52.
- BAYSAL, O.; HOLMES, R.; GODFREY, M. W. Developer dashboards: The need for qualitative analytics. **IEEE Software**, v. 30, n. 4, p. 46–52, jul 2013. ISSN 07407459. Cited on page 31.

BERG, E. A. A Simple Objective Technique for Measuring Flexibility in Thinking. **The Journal of General Psychology**, Taylor & Francis, v. 39, n. 1, p. 15–22, 1948. ISSN 0022-1309. Cited on page 48.

BERGER, H. Über das Elektrenkephalogramm des Menschen: XII. Mitteilung. **Archiv für Psychiatrie und Nervenkrankheiten**, Springer, v. 106, n. 1, p. 165–187, 1937. ISSN 14338491. Cited on page 48.

BERKA, C. et al. Evaluation of an EEG workload model in an Aegis simulation environment. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. **Defense and security**. [S.l.], 2005. p. 90. ISSN 0277-786X. Cited on page 53.

BIFFI, C. A.; TUISSI, A. **Stato dell'arte sulle tecniche di produzione additiva per metalli**. [S.l.]: Springer Science & Business Media, 2017. v. 109. 5–10 p. ISSN 00260843. ISBN 9788578110796. Cited 2 times in the page 76 and 125.

BOEHM, B. **Software cost estimation with Cocomo II**. [S.l.]: Prentice Hall PTR, 2000. xxxviii, 502 p. ISBN 978-0130266927. Cited on page 38.

BORNAT, R.; DEHNADI, S.; SIMON. Mental models, consistency and programming aptitude. In: AUSTRALIAN COMPUTER SOCIETY, INC. **Conferences in Research and Practice in Information Technology Series**. [S.l.], 2008. v. 78, n. 3, p. 53–61. ISBN 9781920682590. ISSN 14451336. Cited on page 52.

BORSTLER, J.; PAECH, B. The Role of Method Chains and Comments in Software Readability and Comprehension-An Experiment. **IEEE Transactions on Software Engineering**, IEEE, v. 42, n. 9, p. 886–898, 2016. ISSN 00985589. Cited 5 times in the page 31, 34, 38, 58, and 73.

BOTTINI, G. et al. The role of the right hemisphere in the interpretation of figurative aspects of language. **Brain**, Oxford Univ Press, v. 117, n. 6, p. 1241–1253, 1994. Cited on page 48.

BROOKS, R. Towards a theory of the comprehension of computer programs. **International Journal of Man-Machine Studies**, Elsevier, v. 18, n. 6, p. 543–554, 1983. ISSN 00207373. Cited 2 times in the page 39 and 41.

BUDGEN, D.; BRERETON, P. **Performing systematic literature reviews in software engineering**. [S.l.], 2006. v. 45, n. 4ve, 1051 p. Cited 5 times in the page 55, 58, 59, 60, and 61.

BUDGEN, D.; BRERETON, P. **Performing systematic literature reviews in software engineering**. [S.l.], 2006. v. 45, n. 4ve, 1051 p. Cited on page 76.

CAMPBELL, D. T.; RUSSO, M. J. **Social experimentation**. [S.l.]: Sage Publications, Inc, 1999. v. 1. Cited on page 126.

CLARKE, S. **Composition of object-oriented software design models**. Tese (Doutorado) — Dublin City University, 2001. Cited on page 87.

COOPER, N. R. et al. Investigating evoked and induced electroencephalogram activity in task-related alpha power increases during an internally directed attention task. **NeuroReport**, LWW, v. 17, n. 2, p. 205–208, 2006. ISSN 0959-4965. Cited on page 49.

Cooper, R., Osselton, J. W., & Shaw, J. C. **Front Matter**. [S.l.]: Butterworth-Heinemann, 2002. v. 48. i–iv p. ISSN 00115266. ISBN 00251909. Cited on page 50.

COYNE, J. T. et al. Applying Real Time Physiological Measures of Cognitive Load to Improve Training 2 Real Time Physiological Assessment. In: SPRINGER. **Time**. [S.l.], 2009. p. 469–478. Cited on page 53.

CRK, I.; KLUTHE, T.; STEFIK, A. Understanding Programming Expertise. **ACM Transactions on Computer-Human Interaction**, v. 23, n. 1, p. 1–29, 2015. ISSN 10730516. Cited 5 times in the page 31, 49, 50, 52, and 74.

CRK, I.; KLUTHE, T.; STEFIK, A. Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes. **ACM Transactions on Computer-Human Interaction (TOCHI)**, ACM, New York, NY, USA, v. 23, n. 1, p. 2, dec 2015. ISSN 1073-0516. Cited 3 times in the page 32, 74, and 75.

CURTIS, B. et al. Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. **IEEE Transactions on software engineering**, IEEE, n. 2, p. 96–104, 1979. Cited on page 101.

DASHOFY, E. M.; HOEK, A. van der; TAYLOR, R. N. A comprehensive approach for the development of modular software architecture description languages. **ACM Transactions on Software Engineering and Methodology**, ACM, v. 14, n. 2, p. 199–245, 2005. ISSN 1049331X. Cited on page 34.

DAVIES, S. P. The nature and development of programming plans. **International Journal of Man-Machine Studies**, Elsevier, v. 32, n. 4, p. 461–481, 1990. Cited on page 39.

DELORME, A.; MAKEIG, S. EEGLAB: An open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. **Journal of Neuroscience Methods**, Elsevier, v. 134, n. 1, p. 9–21, 2004. ISSN 01650270. Cited on page 53.

DIETRICH, J. et al. Barriers to modularity-an empirical study to assess the potential for modularisation of java programs. In: SPRINGER. **International Conference on the Quality of Software Architectures**. [S.l.], 2010. p. 135–150. Cited on page 95.

DIJKSTRA, E. W. How do we tell truths that might hurt? BT - Selected Writings on Computing: A Personal Perspective. In: **Selected Writings on Computing: A Personal Perspective**. [S.l.]: Springer, 1982. p. 129. Cited on page 48.

EHOW. Factors Affecting Computer program. **Digital Repository Iowa State University**, <http://lib.dr.iastate.edu/>, 2014. Cited on page 40.

EKMAN, P.; LEVENSON, R.; FRIESEN, W. Autonomic nervous system activity distinguishes among emotions. **Science**, American Association for the Advancement of Science, v. 221, n. 4616, p. 1208–1210, 1983. ISSN 0036-8075. Cited on page 46.

Emotiv Systems. **Emotiv EPOC**. 2017. Disponível em: <<http://www.emotiv.com>>. Cited 2 times in the page 49 and 66.

ENGLE, R. W. Working memory capacity as executive attention. **Current Directions in Psychological Science**, SAGE Publications Sage CA: Los Angeles, CA, v. 11, n. 1, p. 19–23, 2002. ISSN 09637214. Cited on page 52.

ERICSSON, K. A.; SIMON, H. A. **Protocol Analysis**. [S.l.]: MIT press Cambridge, MA, 1984. 426 p. Cited on page 40.

ERLIKH, L. Leveraging legacy system dollars for e-business. **IT Professional**, IEEE, v. 2, n. 3, p. 17–23, 2000. ISSN 15209202. Cited on page 38.

ESTEVAJ.C.; REYNOLDSR.G. Identifying reusable software components by induction. **International Journal of Software Engineering and Knowledge Engineering**, World Scientific, v. 1, n. 3, p. 271–292, 1991. Cited on page 43.

EVERS, K.; SIGMAN, M. Possibilities and limits of mind-reading: A neurophilosophical perspective. **Consciousness and Cognition**, v. 22, n. 3, p. 887–897, 2013. ISSN 10902376. Cited on page 75.

FARIAS, K.; GARCIA, A.; LUCENA, C. Effects of stability on model composition effort: an exploratory study. **Software & Systems Modeling**, Springer, v. 13, n. 4, p. 1473–1494, 2014. Cited 2 times in the page 95 and 125.

FARIAS, K. et al. Evaluating the effort of composing design models: a controlled experiment. **Software and Systems Modeling**, v. 14, n. 4, p. 1349–1365, 2015. ISSN 16191374. Cited on page 57.

FARIAS, K. et al. Evaluating the effort of composing design models: a controlled experiment. **Software & Systems Modeling**, Springer, v. 14, n. 4, p. 1349–1365, 2015. Cited 2 times in the page 95 and 125.

FARIAS, K. et al. Toward an architecture for model composition techniques. In: **SEKE**. [S.l.: s.n.], 2015. p. 656–659. Cited on page 87.

FERNÁNDEZ-SÁEZ, A. M.; GENERO, M.; CHAUDRON, M. R. Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. **Information and Software Technology**, v. 55, n. 7, p. 1119–1142, 2013. ISSN 0950-5849. Cited 3 times in the page 60, 61, and 63.

FERNÁNDEZ-SÁEZ, A. M.; GENERO, M.; CHAUDRON, M. R. V. Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study. **Information and Software Technology**, Elsevier, v. 55, n. 7, p. 1119–1142, 2013. ISSN 0950-5849. Cited on page 76.

FIEBACH, C. J. et al. Revisiting the role of Broca's area in sentence processing: Syntactic integration versus syntactic working memory. **Human Brain Mapping**, Wiley Online Library, v. 24, n. 2, p. 79–91, 2005. ISSN 10659471. Cited on page 48.

FOSTER, J. R. **Cost factors in software maintenance**. Tese (Doutorado) — Durham University, 1993. Cited 2 times in the page 31 and 38.

FOX, M. C.; ERICSSON, K. A.; BEST, R. Do procedures for verbal reporting of thinking have to be reactive? A meta-analysis and recommendations for best reporting methods. **Psychological Bulletin**, American Psychological Association, v. 137, n. 2, p. 316–344, 2011. ISSN 1939-1455. Cited on page 40.

FRISTON, K. et al. Psychophysiological and Modulatory Interactions in Neuroimaging. **NeuroImage**, Elsevier, v. 6, n. 3, p. 218–229, 1997. ISSN 10538119. Cited on page 37.

- FRITZ, T. et al. Using psycho-physiological measures to assess task difficulty in software development. In: ACM. **Proceedings of the 36th International Conference on Software Engineering**. [S.l.], 2014. p. 402–413. Cited on page 40.
- FRITZ, T.; MÜLLER, S. C. Leveraging biometric data to boost software developer productivity. In: IEEE. **Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on**. [S.l.], 2016. v. 5, p. 66–77. Cited 6 times in the page 32, 44, 45, 46, 74, and 75.
- GALÁN, F. et al. A brain-actuated wheelchair: Asynchronous and non-invasive Brain-computer interfaces for continuous control of robots. **Clinical Neurophysiology**, Elsevier, v. 119, n. 9, p. 2159–2169, 2008. ISSN 13882457. Cited on page 37.
- GEVINS, A. et al. High-resolution EEG mapping of cortical activation related to working memory: Effects of task difficulty, type of processing, and practice. **Cerebral Cortex**, Oxford Univ Press, v. 7, n. 4, p. 374–385, 1997. ISSN 10473211. Cited on page 52.
- GILMORE, D. J.; GREEN, T. R. G. Programming plans and programming expertise. **The Quarterly Journal of Experimental Psychology Section A**, Taylor & Francis, v. 40, n. 3, p. 423–442, 1988. ISSN 0272-4987. Cited on page 39.
- GRABNER, R. H.; STERN, E.; NEUBAUER, A. C. When intelligence loses its impact: Neural efficiency during reasoning in a familiar area. **International Journal of Psychophysiology**, Elsevier, v. 49, n. 2, p. 89–98, 2003. ISSN 01678760. Cited on page 51.
- GREEN, T. R. G.; PETRE, M. Usability analysis of visual programming environments. **Journal of Visual Languages and Computing**, Elsevier, v. 7, n. 2, p. 131–174, 1996. Cited on page 74.
- GRIMES, D. et al. Feasibility and pragmatics of classifying working memory load with an electroencephalograph. In: ACM. **Proceeding of the twenty-sixth annual CHI conference on Human factors in computing systems - CHI '08**. [S.l.], 2008. p. 835. ISBN 9781605580111. Cited on page 53.
- GRODZINSKY, Y.; SANTI, A. The battle for Broca's region. **Trends in cognitive sciences**, Elsevier, v. 12, n. 12, p. 474–480, 2008. Cited on page 48.
- GUEVARA, F. González-Ladrón-de; FERNÁNDEZ-DIEGO, M.; LOKAN, C. The usage of ISBSG data fields in software effort estimation: A systematic mapping study. **Journal of Systems and Software**, v. 113, p. 188–215, 2016. ISSN 01641212. Cited on page 56.
- GURUMURTHY, S.; MAHIT, V. S.; GHOSH, R. Analysis and simulation of brain signal data by EEG signal processing technique using. **International Journal of Engineering and Technology**, v. 5, n. 3, p. 2771–2776, 2013. ISSN 09754024. Cited on page 53.
- HANNEMANN, J.; KICZALES, G. Design pattern implementation in java and aspectj. In: ACM. **ACM Sigplan Notices**. [S.l.], 2002. v. 37, n. 11, p. 161–173. Cited on page 95.
- He Huang et al. Interactive Multimodal Biofeedback for Task-Oriented Neural Rehabilitation. In: IEEE. **2005 IEEE Engineering in Medicine and Biology 27th Annual Conference**. [S.l.], 2005. p. 2547–2550. ISBN 0-7803-8741-4. ISSN 1557-170X. Cited on page 44.



HYMAN, R. Quasi-experimentation: Design and analysis issues for field settings (book). **Journal of Personality Assessment**, Taylor & Francis, v. 46, n. 1, p. 96–97, 1982. Cited on page 126.

IQBAL, S. T.; BAILEY, B. P. Investigating the effectiveness of mental workload as a predictor of opportune moments for interruption. In: ACM. **CHI '05 extended abstracts on Human factors in computing systems - CHI '05**. [S.l.], 2005. p. 1489. ISBN 1595930027. ISSN 1595930027. Cited on page 45.

IWASA, Y. Case studies in superconducting magnets: Design and operational issues: Second edition. **Case Studies in Superconducting Magnets: Design and Operational Issues: Second Edition**, IEEE, v. 100, n. 1, p. 1–682, 2009. ISSN 00028282. Cited on page 38.

JAIN, A.; HONG, L.; PANKANTI, S. Biometric identification. **Communications of the ACM**, ACM, v. 43, n. 2, p. 90–98, 2000. ISSN 00010782. Cited on page 45.

JELLIFFE. **Vergleichende Lokalisationslehre der Grosshirnrinde**. [S.l.]: World Scientific, 1910. v. 37. 783–784 p. ISSN 0022-3018. ISBN 0387269177. Cited on page 47.

JELLIFFE. **Vergleichende Lokalisationslehre der Grosshirnrinde**. [S.l.]: Springer, 1910. 783–784 p. Cited on page 47.

JENSEN, O. Oscillations in the Alpha Band (9-12 Hz) Increase with Memory Load during Retention in a Short-term Memory Task. **Cerebral Cortex**, Oxford Univ Press, v. 12, n. 8, p. 877–882, 2002. ISSN 14602199. Cited on page 49.

JORGENSEN, M. Practical guidelines for expert-judgment-based software effort estimation. **IEEE software**, IEEE, v. 22, n. 3, p. 57–63, 2005. Cited on page 125.

JULNES, G. **Review of Experimental and Quasi-experimental Designs for Generalized Causal Inference**. [S.l.]: Houghton, Mifflin and Company, 2004. v. 27. 173–185 p. ISSN 01497189. ISBN 0395615569. Cited on page 76.

KAUFMAN, L. et al. Modulation of spontaneous brain activity during mental imagery. **Journal of Cognitive Neuroscience**, MIT Press, v. 2, n. 2, p. 124–132, 1990. ISSN 0898-929X. Cited on page 50.

KELLEHER, C.; PAUSCH, R. Lowering the barriers to programming. **ACM Computing Surveys**, ACM, v. 37, n. 2, p. 83–137, 2005. ISSN 03600300. Cited on page 31.

KITCHENHAM, B.; BUDGEN, D.; BRERETON, O. P. The value of mapping studies – A participant-observer case study. In: **Proceedings of the international conference on Evaluation and Assessment in Software Engineering (EASE'10)**. Swinton, UK, UK: British Computer Society, 2010. (EASE'10), p. 25–33. Cited on page 63.

KITCHENHAM, B.; BUDGEN, D.; BRERETON, O. P. The value of mapping studies – A participant-observer case study. In: **Proceedings of the international conference on Evaluation and Assessment in Software Engineering (EASE'10)**. [S.l.: s.n.], 2010. v. 14, p. 25–33. Cited on page 76.

KITCHENHAM, B. A.; BUDGEN, D.; Pearl Brereton, O. Using mapping studies as the basis for further research - A participant-observer case study. **Information and Software Technology**, Butterworth-Heinemann, Newton, MA, USA, v. 53, n. 6, p. 638–651, jun 2011. ISSN 09505849. Cited 2 times in the page 55 and 61.

KITCHENHAM, B. A.; BUDGEN, D.; Pearl Brereton, O. Using mapping studies as the basis for further research - A participant-observer case study. **Information and Software Technology**, Elsevier, v. 53, n. 6, p. 638–651, 2011. ISSN 09505849. Cited on page 76.

KLIMESCH, W. Memory processes brain oscillations and EEG synchronization Memory processes , brain oscillations and EEG synchronization. **International Journal of Psychophysiology**, Elsevier, v. 43, n. 1-2, p. 61–100, 1996. Cited 2 times in the page 49 and 50.

KLIMESCH, W. EEG alpha and theta oscillations reflect cognitive and memory performance: A review and analysis. **Brain Research Reviews**, Elsevier, v. 29, n. 2-3, p. 169–195, 1999. ISSN 01650173. Cited 4 times in the page 49, 50, 51, and 53.

KLIMESCH, W.; SAUSENG, P.; GERLOFF, C. Enhancing cognitive performance with repetitive transcranial magnetic stimulation at human individual alpha frequency. **European Journal of Neuroscience**, Wiley Online Library, v. 17, n. 5, p. 1129–1133, 2003. ISSN 0953816X. Cited on page 51.

KLIMESCH, W.; SAUSENG, P.; HANSLMAYR, S. EEG alpha oscillations: The inhibition-timing hypothesis. **Brain Research Reviews**, Elsevier, v. 53, n. 1, p. 63–88, 2007. ISSN 01650173. Cited 3 times in the page 49, 50, and 51.

KLINGNER, J.; TVERSKY, B.; HANRAHAN, P. Effects of visual and verbal presentation on cognitive load in vigilance, memory, and arithmetic tasks. **Psychophysiology**, Wiley Online Library, v. 48, n. 3, p. 323–332, 2011. ISSN 0048-5772 SRC - GoogleScholar FG - 0. Cited on page 38.

KO, A. J.; MYERS, B. A. A framework and methodology for studying the causes of software errors in programming systems. **Journal of Visual Languages and Computing**, Elsevier, v. 16, n. 1-2 SPEC. ISS., p. 41–84, 2005. ISSN 1045926X. Cited on page 45.

KOENEMANN, J.; ROBERTSON, S. P. Expert problem solving strategies for program comprehension. In: ACM. **Conference on Human factors In computing systems (CHI)**. [S.l.], 1991. p. 125–130. ISBN 0897913833. Cited on page 41.

KOZACZYNSKI, W.; NING, J.; ENGBERTS, A. Program Concept Recognition and Transformation. **IEEE Transactions on Software Engineering**, IEEE, v. 18, n. 12, p. 1065–1075, 1992. ISSN 00985589. Cited on page 38.

KRUCHTEN, P. Architectural blueprints—the “4+ 1” view model of software architecture. **Tutorial Proceedings of Tri-Ada**, v. 95, p. 540–555, 1995. Cited on page 87.

KRUCHTEN, P. B. The 4+ 1 view model of architecture. **IEEE software**, IEEE, v. 12, n. 6, p. 42–50, 1995. Cited on page 86.

KUZNETSOV, N. A. et al. Effect of precision aiming on respiration and the postural-respiratory synergy. **Neuroscience Letters**, Elsevier, v. 502, n. 1, p. 13–17, 2011. ISSN 03043940. Cited on page 46.

LEHMAN, M. M.; RAMIL, J. F. Software evolution - Background, theory, practice. **Information Processing Letters**, Elsevier, v. 88, n. 1-2, p. 33–44, 2003. ISSN 00200190. Cited 2 times in the page 42 and 43.



- LETHBRIDGE, T. C.; SINGER, J.; FORWARD, A. Use Documentation : The State of the Practice Documentation. **Ieee Focus**, IEEE, v. 20, n. 6, p. 5, 2003. Cited on page 31.
- LI, M.; LU, B.-L. Emotion Classification Based on Gamma-band EEG. In: IEEE. **Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE**. [S.l.], 2009. p. 1–4. ISBN 9781424432967. Cited on page 46.
- LIU, J. et al. Cognitive pilot-aircraft interface for single-pilot operations. **Knowledge-Based Systems**, Elsevier, v. 112, p. 37–53, 2016. ISSN 09507051. Cited on page 44.
- MACARTHUR, D. W. et al. **Modular Design in Treaty Verification Equipment**. [S.l.], 2015. Cited on page 117.
- MALMIVUO, J.; PLONSEY, R. **Bioelectromagnetism: Principles and Applications of Bioelectric and Biomagnetic Fields**. [S.l.]: Oxford University Press, USA, 2012. 1–506 p. ISSN 15204804. ISBN 9780199847839. Cited 2 times in the page 50 and 51.
- MATHEWSON, K. E. et al. Different slopes for different folks: Alpha and delta EEG power predict subsequent video game learning rate and improvements in cognitive control tasks. **Psychophysiology**, Wiley Online Library, v. 49, n. 12, p. 1558–1570, 2012. ISSN 00485772. Cited 2 times in the page 49 and 53.
- MAYRHAUSER, A. V.; VANS, A. M. Program comprehension during software maintenance and evolution. **Computer**, IEEE, v. 28, n. 8, p. 44–55, 1995. ISSN 00189162. Cited 5 times in the page 32, 37, 38, 39, and 41.
- MAYRHAUSER, A. von; VANS, A. Comprehension Processes During Large Research Paper Scale Maintenance. In: IEEE COMPUTER SOCIETY PRESS. **Proceedings of the 16th International Conference on Software Engineering**. [S.l.], 1994. p. 39–48. Cited 3 times in the page 41, 42, and 52.
- MAYRHAUSER, A. von; VANS, A. M. Program understanding behavior during adaptation of large scale software. In: ACM. **6th International Workshop on Program Comprehension**. [S.l.], 1998. p. 164–172. ISBN 1092-8138 0-8186-8560-3. ISSN 1092-8138. Cited 2 times in the page 31 and 38.
- MCCABE, T. J. A complexity measure. **IEEE Transactions on software Engineering**, IEEE, n. 4, p. 308–320, 1976. Cited on page 101.
- MCCARTHY, G.; WOOD, C. Scalp distributions of event related potentials: an ambiguity associated with analysis of variance models. **Electroencephalography and Clinical Neurophysiology**, Elsevier, v. 61, n. 3, p. 226–227, 1985. Cited on page 48.
- MENS, T. A state-of-the-art survey on software merging. **IEEE Transactions on Software Engineering**, IEEE, v. 28, n. 5, p. 449–462, 2002. ISSN 00985589. Cited on page 76.
- MEYER, B. **Object-Oriented Software Construction**. [S.l.]: Prentice hall New York, 1997. v. 2. 1296 p. ISBN 978-0136291558. Cited on page 43.
- MIARA, R. J. et al. Program Indentation and Comprehensibility. **Communications of the ACM**, ACM, v. 26, n. 11, p. 861–867, 1983. ISSN 00010782. Cited on page 43.
- MITCHELL, M. L.; JOLLEY, J. M. **Research design explained**. [S.l.]: Cengage Learning, 2012. Cited on page 127.

MOHA, N. et al. Decor: A method for the specification and detection of code and design smells. **IEEE Transactions on Software Engineering**, IEEE, v. 36, n. 1, p. 20–36, 2010. Cited on page 96.

MÜLLER, S. C.; FRITZ, T. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In: IEEE. **Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on**. [S.l.], 2015. v. 1, p. 688–699. Cited on page 47.

MÜLLER, S. C.; FRITZ, T. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. **Proceedings - International Conference on Software Engineering**, v. 1, p. 688–699, 2015. ISSN 02705257. Cited 2 times in the page 55 and 74.

MÜLLER, S. C.; FRITZ, T. Using (bio)metrics to predict code quality online. In: ACM. **Proceedings of the 38th International Conference on Software Engineering - ICSE '16**. [S.l.], 2016. p. 452–463. ISBN 9781450339001. ISSN 02705257. Cited on page 47.

MUNRO, M. J. Product metrics for automatic identification of " bad smell" design problems in java source-code. In: IEEE. **Software Metrics, 2005. 11th IEEE International Symposium**. [S.l.], 2005. p. 15–15. Cited on page 96.

MURUGAPPAN, M.; NAGARAJAN, R.; YAACOB, S. Modified Energy Based Time-Frequency Features for Classifying Human Emotions using EEG. **the International Conference on Man-Machine Systems (ICoMMS)**, n. October, p. 11–13, 2009. Cited on page 45.

MURUGAPPAN, M. et al. EEG feature extraction for classifying emotions using FCM and FKM. **International Journal of Computers and Communications**, v. 1, n. 2, p. 21–25, 2007. Cited 3 times in the page 44, 45, and 47.

Neurosky. 2016. Cited on page 66.

NEWMAN, P. S. Towards an integrated development environment. **IBM Systems Journal**, IBM, v. 21, n. 1, p. 81–107, 1982. ISSN 0018-8670. Cited on page 43.

NGUYEN, V. Improved Size and Effort Estimation Models for Software Maintenance. In: IEEE. **Software Maintenance (ICSM), 2010 IEEE International Conference on**. [S.l.], 2010. p. 237. ISBN 9781424486281. ISSN 1063-6773. Cited 2 times in the page 31 and 38.

NGUYEN, V.; BOEHM, B.; DANPHITSANUPHAN, P. A controlled experiment in assessing and estimating software maintenance tasks. **Information and Software Technology**, Elsevier, v. 53, n. 6, p. 682–691, 2011. ISSN 09505849. Cited on page 31.

NICOLAS-ALONSO, L. F.; GOMEZ-GIL, J. Brain computer interfaces, a review. **Sensors**, Molecular Diversity Preservation International, v. 12, n. 2, p. 1211–1279, 2012. Cited on page 55.

NIJBOER, F. et al. An auditory brain-computer interface (BCI). **Journal of Neuroscience Methods**, Elsevier, v. 167, n. 1, p. 43–50, 2008. ISSN 01650270. Cited on page 48.

NUNEZ, P. L.; SRINIVASAN, R. **Electric Fields of the Brain: The Neurophysics of EEG, 2nd Edition: 9780195050387: Medicine & Health Science Books@ Amazon.com**. [S.l.]: Oxford University Press, USA, 2006. 640 p. ISBN 978-0-19-505038-7. Cited 2 times in the page 33 and 50.

O'BRIEN, M. P. Software comprehension—a review & research direction. **Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report**, 2003. Cited 2 times in the page 39 and 42.

OLIVEIRA, K.; BREITMAN, K.; OLIVEIRA, T. Ontology aided model comparison. In: IEEE. **Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on**. [S.l.], 2009. p. 78–83. Cited on page 87.

ORIOLO, M.; MARCO, J.; FRANCH, X. Quality models for web services: A systematic mapping. **Information and Software Technology**, Elsevier, v. 56, n. 10, p. 1167–1182, 2014. ISSN 09505849. Cited on page 76.

PAAS, F. et al. Cognitive Load Measurement as a Means to Advance Cognitive Load Theory. **Educational Psychologist**, Taylor & Francis, v. 1520, n. 38, p. 43–52, 2010. ISSN 0046-1520. Cited on page 45.

PACE-SCHOTT, E. F.; HOBSON, J. A. The Neurobiology of Sleep: Genetics, cellular physiology and subcortical networks. **Nature Reviews Neuroscience**, Nature Publishing Group, v. 3, n. 8, p. 591–605, 2002. ISSN 1471-003X. Cited on page 49.

PENNINGTON, N. Comprehension strategies in programming. In: ABLEX PUBLISHING CORP. **Empirical studies of programmers: second workshop**. [S.l.], 1987. p. 100–111. ISBN 0893914614. Cited on page 41.

PENNINGTON, N. Stimulus structures and mental representations in expert comprehension of computer programs. **Cognitive Psychology**, Elsevier, v. 19, n. 3, p. 295–341, 1987. ISSN 00100285. Cited 4 times in the page 40, 41, 42, and 52.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, v. 64, p. 1–18, 2015. ISSN 0950-5849. Cited 5 times in the page 36, 55, 58, 60, and 61.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: An update. **Information and Software Technology**, Elsevier, v. 64, p. 1–18, 2015. ISSN 0950-5849. Cited on page 76.

PETERSON, K. **Implementing Lean and Agile Software Development in Industry**. 309 p. Tese (Doutorado) — Blekinge Institute of Technology, 2010. Cited on page 31.

PETERSSON, K. M.; FOLIA, V.; HAGOORT, P. What artificial grammar learning reveals about the neurobiology of syntax. **Brain and Language**, Elsevier, v. 120, n. 2, p. 83–95, 2012. ISSN 0093934X. Cited on page 48.

PETRUCCELLI, J. D.; NANDRAM, B.; CHEN, M. **Applied statistics for engineers and scientists**. [S.l.]: Prentice Hall New Jersey, 1999. Cited on page 126.

PFURTSCHHELLER, G.; ARANIBAR, A. Event related cortical desynchronization detected by power measurements of scalp EEG. **Electroencephalography and Clinical Neurophysiology**, Elsevier, v. 42, n. 5, p. 817–826, 1977. Cited on page 50.

PICARD, R. W.; VYZAS, E.; HEALEY, J. Toward machine emotional intelligence: Analysis of affective physiological state. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, IEEE, v. 23, n. 10, p. 1175–1191, 2001. ISSN 01628828. Cited on page 46.

PRICE, C. J. et al. Brain activity during reading. The effect of exposure duration and task. **Brain**, Oxford Univ Press, v. 117, n. 6, p. 1255–1269, 1994. Cited on page 48.

QIU, D. et al. Regression Testing of Web Service: A Systematic Mapping Study. **ACM Computing Surveys**, ACM, New York, NY, USA, v. 47, n. 2, p. 1–46, aug 2014. ISSN 03600300. Cited 4 times in the page 59, 60, 63, and 76.

RAJLICH, V. Intensions are a key to program comprehension. In: IEEE. **IEEE International Conference on Program Comprehension**. [S.l.], 2009. p. 1–9. ISBN 9781424439973. ISSN 1063-6897. Cited 2 times in the page 37 and 38.

RAMBALLY, G. K. **The influence of color on program readability and comprehensibility**. [S.l.]: ACM, 1986. v. 18. ISSN 00978418. Cited on page 43.

RAWASSIZADEH, R.; PRICE, B. A.; PETRE, M. Wearables. **Communications of the ACM**, ACM, v. 58, n. 1, p. 45–47, 2014. ISSN 00010782. Cited on page 47.

RAY, W.; COLE, H. EEG alpha activity reflects attentional demands, and beta activity reflects emotional and cognitive processes. **Science**, v. 228, n. 4700, p. 750–752, 1985. ISSN 0036-8075. Cited on page 49.

RAZALI, N. M.; WAH, Y. B. et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. **Journal of statistical modeling and analytics**, v. 2, n. 1, p. 21–33, 2011. Cited on page 111.

RICCA, F. et al. How developers' experience and ability influence web application comprehension tasks supported by uml stereotypes: A series of four experiments. **IEEE Transactions on Software Engineering**, IEEE, v. 36, n. 1, p. 96–118, 2010. Cited on page 96.

RICHTER, P. et al. Psychophysiological analysis of mental load during driving on rural roads-a quasi-experimental field study. **Ergonomics**, Taylor & Francis, v. 41, n. 5, p. 593–609, 1998. ISSN 0014-0139. Cited 2 times in the page 44 and 46.

RIHS, T. A.; MICHEL, C. M.; THUT, G. Mechanisms of selective inhibition in visual spatial attention are indexed by alpha-band EEG synchronization. **European Journal of Neuroscience**, Wiley Online Library, v. 25, n. 2, p. 603–610, 2007. ISSN 0953816X. Cited 2 times in the page 49 and 50.

RIST, R. Plans in Programming : Definition, Demonstration and Development. In: **Empirical Software Engineering**. [S.l.: s.n.], 1987. p. 28–45. ISBN 0-89391-388-X. Cited on page 39.

ROEHM, T. et al. How Do Professional Developers Comprehend Software? In: IEEE PRESS. **Proceedings of the 34th International Conference on Software Engineering**. [S.l.], 2012. p. 255–265. ISBN 978-1-4673-1067-3. Cited on page 40.

ROZENBLIT, J. W. Cognitive Computing: Principles, Architectures, and Applications. In: **Proceedings of the 19th European Conference on Modelling and Simulation (ECMS)**. [S.l.: s.n.], 2005. Cited 2 times in the page 55 and 56.

SALOMON, D. **Assemblers and Loaders**. [S.l.: s.n.], 1993. ISBN 0130525642. Cited on page 42.

SCHAPIRO, S. B.; HENRY, M. H. Engineering agile systems through architectural modularity. In: IEEE. **Systems Conference (SysCon), 2012 IEEE International**. [S.l.], 2012. p. 1–6. Cited on page 117.

SCHOOLER, J. W. Introspecting in the spirit of William James: Comment on Fox, Ericsson, and Best (2011). **Psychological Bulletin**, American Psychological Association, v. 137, n. 2, p. 345–350, 2011. ISSN 1939-1455. Cited on page 40.

SHADISH, W. R.; COOK, T. D.; CAMPBELL, D. T. **Experimental and quasi-experimental designs for generalized causal inference**. [S.l.]: Wadsworth Cengage learning, 2002. Cited on page 127.

SHAFT, T. M.; VESSEY, I. Research Report—The Relevance of Application Domain Knowledge: The Case of Computer Program Comprehension. **Information Systems Research**, INFORMS, v. 6, n. 3, p. 286–299, 1995. ISSN 1047-7047. Cited 2 times in the page 40 and 42.

SHARIT, J.; SALVENDY, G. **Handbook of Human Factors and Ergonomics**. [S.l.]: John Wiley & Sons, 2006. 708 p. ISSN 10908471. ISBN 9780470048207. Cited on page 44.

SHIFFRIN, R. M.; SCHNEIDER, W. Controlled and automatic human information processing: II. Perceptual learning, automatic attending and a general theory. **Psychological Review**, American Psychological Association, v. 84, n. 2, p. 127–190, 1977. ISSN 0033-295X. Cited on page 52.

SHNEIDERMAN, B. Exploratory experiments in programmer behavior. **International Journal of Computer & Information Sciences**, Springer, v. 5, n. 2, p. 123–143, 1976. ISSN 00917036. Cited 4 times in the page 32, 39, 40, and 42.

SHNEIDERMAN, B.; BEN. Software psychology : human factors in computer and information systems. **Winthrop Publishers**, p. 320, 1980. Cited on page 40.

SIEGMUND, J. Program comprehension: Past, present, and future. In: **IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)**. [S.l.: s.n.], 2016. v. 5, p. 13–20. Cited 13 times in the page 11, 31, 32, 33, 34, 37, 38, 39, 40, 42, 43, 74, and 75.

SIEGMUND, J. et al. Understanding understanding source code with functional magnetic resonance imaging. In: ACM. **Proceedings of the 36th International Conference on Software Engineering - ICSE 2014**. [S.l.], 2014. p. 378–389. ISBN 9781450327565. ISSN 02705257. Cited 6 times in the page 45, 47, 48, 73, 74, and 75.

SIEGMUND, J. et al. Measuring and modeling programming experience. **Empirical Software Engineering**, Springer, v. 19, n. 5, p. 1299–1334, 2014. ISSN 15737616. Cited on page 52.

SIEGMUND, J.; SCHUMANN, J. Confounding parameters on program comprehension: a literature survey. **Empirical Software Engineering**, Springer, v. 20, n. 4, p. 1159–1192, 2014. ISSN 15737616. Cited on page 37.

SJØGAARD, G.; LUNDBERG, U.; KADEFORS, R. The role of muscle activity and mental load in the development of pain and degenerative processes at the muscle cell level during computer work. **European Journal of Applied Physiology**, Springer, v. 83, n. 2-3, p. 99–105, 2000. ISSN 14396319. Cited on page 44.



SKOSNIK, P. et al. Neural Correlates of Artificial Grammar Learning. **NeuroImage**, Elsevier, v. 17, n. 3, p. 1306–1314, 2002. ISSN 10538119. Cited on page 48.

ŠMITE, D. et al. Empirical evidence in global software engineering: A systematic review. **Empirical Software Engineering**, Springer US, v. 15, n. 1, p. 91–118, 2010. ISSN 13823256. Cited on page 61.

SOLOWAY, E.; ADELSON, B.; EHRLICH, K. Knowledge and Processes in the Comprehension of Computer Programs. **The Nature of Expertise**, Lawrence Erlbaum Associates Hillsdale, NJ, p. 129–152, 1988. Cited on page 41.

SOLOWAY, E.; EHRLICH, K. Empirical Studies of Programming Knowledge.pdf. **IEEE Transactions on software engineering**, IEEE, n. 5, p. 595–609, 1984. Cited 5 times in the page 32, 39, 40, 41, and 43.

SPINELLIS, D. Code documentation. **IEEE Software**, IEEE, v. 27, n. 4, p. 18–19, 2010. ISSN 07407459. Cited on page 31.

SPRINGER. Yes: Using tutor and sensor data to predict moments of delight during instructional activities. In: **User Modeling, Adaptation, and Personalization, Proceedings**. [S.l.], 2010. v. 6075, p. 159–170. ISBN 978-3-642-13469-2. ISSN 0302-9743. Cited on page 46.

STIPACEK, A. et al. Sensitivity of human EEG alpha band desynchronization to different working memory components and increasing levels of memory load. **Neuroscience Letters**, Elsevier, v. 353, n. 3, p. 193–196, 2003. ISSN 03043940. Cited on page 51.

STOREY, M.-A. Theories, Methods and Tools in Program Comprehension: Past, Present and Future. In: IEEE. **13th International Workshop on Program Comprehension (IWPC'05)**. [S.l.], 2005. p. 181–191. ISBN 0-7695-2254-8. Cited 2 times in the page 38 and 43.

SWAIN, R. Psychophysiological responses to changes in workload during simulated air traffic control. **Biological psychology**, Elsevier, v. 0511, n. 95, p. 361–377, 1996. Cited on page 44.

SWELLER, J. Implications of Cognitive Load Theory for Multimedia Learning. **The Cambridge Handbook of Multimedia Learning**, Cambridge University Press Cambridge, v. 27, p. 19–30, 2014. ISSN 0888-4080. Cited on page 52.

SWELLER, J.; AYRES, P.; KALYUGA, S. **Cognitive Load Theory**. [S.l.]: Cambridge University Press, 2011. ISSN 00797421. ISBN 978-1-4419-8125-7. Cited on page 45.

THUNG, F. et al. Popularity , Interoperability , and Impact of Programming Languages in 100 , 000 Open Source Projects ere To cite this version : . In: IEEE. **37th Annual International Computer Software & Applications Conference (COMP-SAC)**. [S.l.], 2013. p. 1–10. Cited on page 33.

TROCHIM, W. M.; DONNELLY, J. P. Research methods knowledge base. Atomic Dog Pub., 2001. Cited on page 126.

VANDENBERGHE, R. et al. Attention to one or two features in left or right visual field: a positron emission tomography study. **Journal of Neuroscience**, Soc Neuroscience, v. 17, n. 10, p. 3739–50, 1997. ISSN 0270-6474. Cited on page 48.

VENABLES, P.; CHRISTIE, M. **Electrodermal Activity**. [S.l.]: Springer Science & Business Media, 1980. 3–67 p. ISBN 978-1-4614-1125-3. Cited on page 46.

Virtual Machinery. **Object-Oriented Software Metrics - Introduction and overview**. 2017. Disponível em: <<http://www.virtualmachinery.com>>. Cited on page 100.

WARD, M. P. Language-oriented programming. **Software-Concepts and Tools**, v. 15, n. 4, p. 147–161, 1994. Cited on page 42.

WELCH, P. D. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Over Short, Modified Periodograms. **IEEE Transactions on Audio and Electroacoustics**, IEEE, v. 15, n. 2, p. 70–73, 1967. ISSN 00189278. Cited on page 50.

WELKER, K. D. The software maintainability index revisited. **CrossTalk**, v. 14, p. 18–21, 2001. Cited on page 101.

WILSON, G. F. An Analysis of Mental Workload in Pilots During Flight Using Multiple Psychophysiological Measures. **The International Journal of Aviation Psychology**, Taylor & Francis, v. 12, n. 1, p. 3–18, 2002. ISSN 1050-8414. Cited on page 44.

WOHLIN, C. et al. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012. Cited 3 times in the page 95, 96, and 127.

WUNDT, W. **Grundzuge der physiologischen Psychologie: By WILHELM WUNDT. 4 Auflage. Leipzig: Wm. Engelmann. 1893**. [S.l.]: W. Engelman, 1895. v. 41. 347–348 p. ISSN 0007-1250. Cited 2 times in the page 32 and 40.

XIA, X. et al. Grey bootstrap method of evaluation of uncertainty in dynamic measurement. **Measurement**, Elsevier, v. 41, n. 6, p. 687–696, 2008. Cited on page 112.

YIP, S.; LAM, T. A software maintenance survey. In: IEEE. **Proceedings of 1st Asia-Pacific Software Engineering Conference**. [S.l.], 1994. p. 70–79. ISBN 0-8186-6960-8. Cited 2 times in the page 31 and 38.

YOUNG, M. S. et al. State of science: mental workload in ergonomics. **Ergonomics**, v. 58, n. 1, p. 1–17, 2015. ISSN 0014-0139. Cited on page 46.

ZHI, J. et al. Cost, benefits and quality of software development documentation: A systematic mapping. **Journal of Systems and Software**, v. 99, p. 175–198, 2015. ISSN 01641212. Cited on page 56.

ZIGNEGO, M. I. Human Factors in the Design of Naval Vessels. In: SPRINGER. **International Conference on Human-Computer Interaction**. [S.l.], 2014. p. 543–551. Cited on page 55.

ZÜGER, M.; FRITZ, T. Interruptibility of Software Developers and its Prediction Using Psycho-Physiological Sensors. In: ACM. **Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15**. [S.l.], 2015. p. 2981–2990. ISBN 9781450331456. Cited on page 47.



## APPENDIX A – LIST OF PRIMARY STUDIES

Below follow the list of the 12 selected articles in the systematic mapping review.

- S01** Pega Zarjam, Julien Epps, and Nigel H. Lovell. Beyond Subjective Self-Rating: EEG Signal Classification of Cognitive Workload. *IEEE Transactions on Autonomous Mental Development*, v. 7, n. 4, p. 301-310, 2015.
- S02** Martin Spüler, Carina Walter, Wolfgang Rosenstiel, Peter Gerjets, Korbinian Moeller, Elise Klein. EEG-Based prediction of cognitive workload induced by arithmetic: a step towards online adaptation in numerical learning. *ZDM - Mathematics Education*, v. 48, n.3, p. 267–278, 2016.
- S03** Thomas Fritz, Sebastian C. Müller. Leveraging Biometric Data to Boost Software Developer Productivity. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, v. 5, p. 66-77, 2016.
- S04** Sebastian C. Müller. Measuring software developers' perceived difficulty with biometric sensors. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, v. 2, p. 887-890, 2015.
- S05** Elizabeth A. Felton, Justin C. Williams, Gregg C. Vanderheiden and Robert G. Radwin. Mental workload during brain–computer interface training. *Ergonomics*, v. 55, n. 5, p. 526-537, 2012.
- S06** Kathinka Evers, and Mariano Sigman. Possibilities and limits of mind-reading: a neurophilosophical perspective. *Consciousness and cognition*, v. 22, n. 3, p. 887-897, 2013.
- S07** Janet Siegmund. Program Comprehension: Past, Present, and Future. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, v. 5, p. 13-20, 2016.
- S08** Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. Real-time monitoring of neural state in assessing and improving software developers' productivity. In *2015 IEEE 8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, p. 93-96, 2015.
- S09** Mathieu Rodrigue, Jungah Son, Barry Giesbrecht, Matthew Turk and Tobias Höllerer. Spatio-Temporal Detection of Divided Attention in Reading Applications Using EEG and Eye Tracking. In *2015 ACM 20th International Conference on Intelligent User Interfaces*, p. 121-125, 2015.
- S10** Ondrej Polacek and Adam J. Sporcka. Text input for motor-impaired people. *Universal Access in the Information Society*, p. 1-22, 2015.

- S11** Igor Crk and Timothy Kluthe. Toward using alpha and theta brain waves to quantify programmer expertise. In 2014 IEEE 36th Annual International Conference, Engineering in Medicine and Biology Society (EMBC), p. 5373-5376, 2014.
- S12** Igor Crk, Timothy Kluthe and Andreas Stefik. Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes. ACM Transactions on Computer-Human Interaction (TOCHI), v. 23, n. 1, p. 1-29, 2016.

## APPENDIX B – QUALITY ASSESSMENT

Table 28 – Quality Assessment description.

Article	QA1	QA2	QA3	QA4	QA5	QA6	QA7	QA8	QA9	Total
S01	1	1	1	1	0	1	1	1	1	8
S02	1	1	0	1	1	1	1	1	1	8
S03	1	1	1	1	0	1	1	1	1	9
S04	1	1	1	1	1	1	1	1	1	9
S05	1	1	0	1	0	1	1	1	1	7
S06	1	1	0	0	1	1	1	1	1	7
S07	1	1	1	1	1	1	1	1	1	9
S08	1	0	1	1	1	1	1	0	1	7
S09	1	1	1	0	0	1	1	1	1	7
S10	1	1	1	0	1	1	1	1	1	8
S11	1	1	1	0	1	1	1	1	1	8
S12	1	1	1	1	0	1	1	1	1	8

Source: Created by the author



## APPENDIX C – STRING SEARCH ADAPTED BY ELECTRONIC DATABASE

- **ACM (<http://dl.acm.org/>):** ( Electroencephalogram OR EEG OR "Brain-computer interface" OR BCI ) AND ( Understanding OR Comprehension OR "Empirical study" OR "Case study" OR "Controlled Experiment" OR "Source Code" OR "programming language" OR Diagram OR UML )
- **CiteSeerX Library (<http://citeseerx.ist.psu.edu/>):** (( Electroencephalogram AND EEG AND "Brain-computer interface" AND BCI ) AND ( Understanding OR Comprehension ) AND ( "Empirical study" OR "Case study" OR "Controlled Experiment" ) AND ( "Programming Language" OR "Source Code" OR Diagram OR UML ))
- **Google Scholar (<https://scholar.google.com.br/>):** ( Electroencephalogram OR EEG ) ( Brain-computer interface OR BCI ) ( Understanding OR Comprehension ) ( Empirical study OR Case study OR Controlled Experiment ) ( Programming Language OR Source Code OR Diagram OR UML )
- **IEEE Explore (<http://ieeexplore.ieee.org/>):** (Electroencephalogram OR EEG OR Brain-computer interface OR BCI AND Understanding OR Comprehension AND Empirical study OR Case study OR Controlled Experiment AND Programming Language OR Source Code OR Diagram OR UML)
- **Inspec (<http://digital-library.theiet.org/>):** Electroencephalogram OR EEG OR Brain-computer interface OR BCI AND Understanding OR Comprehension OR Empirical study OR Case study OR Controlled Experiment OR Programming Language OR Source Code OR Diagram OR UML
- **Science Direct (<http://www.sciencedirect.com/>):** Electroencephalogram OR EEG AND Brain-computer interface OR BCI AND Understanding OR Comprehension AND Empirical study OR Case study OR Controlled Experiment AND Programming Language OR Source Code OR Diagram OR UML
- **Springer Link (<http://link.springer.com/>):** Electroencephalogram OR EEG AND Brain-computer interface OR BCI AND Understanding OR Comprehension AND Empirical study OR Case study OR Controlled Experiment OR Programming Language OR Source Code OR Diagram OR UML
- **Wiley Online Library (<http://onlinelibrary.wiley.com/>):** ( Electroencephalogram OR EEG ) ( Brain-computer interface OR BCI ) ( Understanding OR Comprehension ) ( Empirical study OR Case study OR Controlled Experiment ) ( Programming Language OR Source Code OR Diagram OR UML )



## APPENDIX D – QUESTIONNAIRE OF CHARACTERISTICS

Figure 43 – Questionnaire - Page I.

## Questionário 1

Tempo:	Hora de Início : ____:____		Hora de Fim : ____:____	
Nome: (opcional)			Óculos:	<input type="checkbox"/> Sim <input type="checkbox"/> Não
			Escrita:	<input type="checkbox"/> Canhoto <input type="checkbox"/> Destro
Idade:		Profissão:		
Cargo Atual:			Empresa:	
Quanto tempo estás no cargo atual?			Sexo:	<input type="checkbox"/> Masc. <input type="checkbox"/> Fem. <input type="checkbox"/> Outro
1) Qual é o seu maior grau de <u>escolaridade</u> ?	<input type="checkbox"/>	Médio - Incompleto		
	<input type="checkbox"/>	Médio - Completo		
	<input type="checkbox"/>	Superior - Incompleto		
	<input type="checkbox"/>	Superior - Completo		
	<input type="checkbox"/>	Pós-graduação (Mestrado) - Incompleto		
	<input type="checkbox"/>	Pós-graduação (Mestrado) - Completo		
	<input type="checkbox"/>	Pós-graduação (Doutorado) - Incompleto		
	<input type="checkbox"/>	Pós-graduação (Doutorado) - Completo		
	Outro: Qual?			
2) Por quanto tempo você <u>estudou</u> (tem estudado) em universidades?	<input type="checkbox"/>	Nunca estudei	<input type="checkbox"/>	6 anos
	<input type="checkbox"/>	1 ano	<input type="checkbox"/>	7 anos
	<input type="checkbox"/>	2 anos	<input type="checkbox"/>	8 anos
	<input type="checkbox"/>	3 anos	<input type="checkbox"/>	9 anos
	<input type="checkbox"/>	4 anos	<input type="checkbox"/>	10 anos
	<input type="checkbox"/>	5 anos	<input type="checkbox"/>	11 anos
	Outro: Qual?			
3) Qual é a sua <u>formação acadêmica?</u> ( em andamento ou concluída )	<input type="checkbox"/>	Sistemas de Informação		
	<input type="checkbox"/>	Ciência da Computação		
	<input type="checkbox"/>	Engenharia de Sistemas		
	<input type="checkbox"/>	Análise de Sistemas		
	<input type="checkbox"/>	Sistemas para Internet		
Outro: Qual?				
4) Quanto tempo você tem de <u>experiência em desenvolvimento</u> de sistemas?	<input type="checkbox"/>	Não tenho	<input type="checkbox"/>	6 anos
	<input type="checkbox"/>	1 ano	<input type="checkbox"/>	7 anos
	<input type="checkbox"/>	2 anos	<input type="checkbox"/>	8 anos
	<input type="checkbox"/>	3 anos	<input type="checkbox"/>	9 anos
	<input type="checkbox"/>	4 anos	<input type="checkbox"/>	10 anos
	<input type="checkbox"/>	5 anos	<input type="checkbox"/>	11 anos
Outro: Qual?				



Figure 44 – Questionnaire - Page II.

Folha Verso

5) Quanto tempo você tem de <b>experiência em modelagem</b> de software?	<input type="checkbox"/>	Não tenho	<input type="checkbox"/>	5 anos
	<input type="checkbox"/>	1 ano	<input type="checkbox"/>	6 anos
	<input type="checkbox"/>	2 anos	<input type="checkbox"/>	7 anos
	<input type="checkbox"/>	3 anos	<input type="checkbox"/>	8 anos
	<input type="checkbox"/>	4 anos	<input type="checkbox"/>	9 anos
	Outro: Qual?			

6) Qual <b>cargo</b> se relaciona melhor com o seu atual conhecimento?	<input type="checkbox"/>	Desenvolvedor de Software
	<input type="checkbox"/>	Arquiteto de Sistemas
	<input type="checkbox"/>	Analista/Desenvolvedor
	<input type="checkbox"/>	Analista de Software
	<input type="checkbox"/>	Gerente de Projetos
	<input type="checkbox"/>	Analista de Negócios
Outro: Qual?		

7) Escolha um <b>nível</b> para o seu <b>conhecimento</b> em cada item listada ao lado, entre <b>1 a 5 (Escala de Likert)</b> ?  1 - Praticamente nenhum conhecimento 2 - Pouco conhecimento 3 - Conhecimento intermediário 4 - Bastante conhecimento 5 - Praticamente todo conhecimento	Item	Escala de Likert				
	JavaScript	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Linguagem R	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Python	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	SQL (ANSI)	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Linguagem C	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Object Oriented	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Linguagem JAVA	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Diagramas UML	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Programação	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Análise de Sistema	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Back-end	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Front-end	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Lógica de Program.	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]
	Língua Inglesa	1 [ ]	2 [ ]	3 [ ]	4 [ ]	5 [ ]

8) Quantas <b>certificações</b> você possui dentro da área da informática?	<input type="checkbox"/>	1 Certificação
	<input type="checkbox"/>	2 Certificações
	<input type="checkbox"/>	3 Certificações
	<input type="checkbox"/>	4 Certificações
	<input type="checkbox"/>	5 Certificações
	<input type="checkbox"/>	Não possuo nenhuma certificação
Outro: Qual?		

Este questionário tem como objetivo o de avaliar a técnica de compreensão proposta pela dissertação do Matheus Segalotto. Desse modo, as respostas para as questões abaixo devem ser baseadas na experiência do participante. O questionário possui duas partes, uma para caracterizar o participante e a outra para coletar as informações. Os participantes não estão sendo avaliados e seus dados não serão divulgados. As questões pessoais acima servem somente para agrupar os dados de acordo com o conhecimento adquirido na compreensão de artefatos de software.

Muito obrigado pela sua participação!

## APPENDIX E – QUESTIONNAIRE OF IMPRESSION

Figure 45 – Questionnaire (After the Experiment).

### Questionário de Pesquisa ( Após o Experimento )

E-mail: \_\_\_\_\_

Nome: \_\_\_\_\_

**Q1 - O tempo foi suficiente para executar as questões do experimento**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---

**Q2 - A tarefa de cada questão estava perfeitamente clara para mim**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---

**Q3 - As questões respondidas nas tarefas estavam perfeitamente claras**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---

**Q4 - Julgue a dificuldade das tarefas relacionadas com a compreensão do código em JAVA**

( ) Muito Baixo ( ) Baixo ( ) Médio ( ) Alto ( ) Muito Alto

---

**Q5 - Julgue a dificuldade das tarefas relacionadas com a pressão para responder**

( ) Muito Baixo ( ) Baixo ( ) Médio ( ) Alto ( ) Muito Alto

---

**Q6 - A fonte do texto estava clara para a compreensão do código-fonte**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---

**Q7 - As opções da tarefa dificultaram a tentativa de adivinhar a resposta correta sem a total compreensão do código-fonte**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---

**Q8 - As instruções do experimento foram claras antes da sua realização**

( ) Discordo Totalmente ( ) Discordo Parcialmente ( ) Indiferente ( ) Concordo Parcialmente ( ) Concordo Totalmente

---



**APPENDIX F – NON-MODULARIZED QUESTIONS**

Figure 46 – Non-Modularized - Question 1.

```
public class Payment {
    public static void main(String[] args) {
        double bill = 15;
        bill = bill - 5;
        bill = bill * 2;
        bill = bill + 10;
        bill = bill / 5;
        System.out.print( ++bill );
    }
}
```

1) 5      2) 7      3) 3      4) 6      5) 8

Source: Created by the author

Figure 47 – Non-Modularized - Question 2.

```
public class Soccer {
    public static void main(String[] args) {
        int total = ( 2 - 1 ) * 2;
        total = total + ( 4 - 1 ) * 2;
        total = total + ( 3 - 1 ) * 2;
        System.out.print( total );
    }
}
```

1) 12      2) 10      3) 14      4) 11      5) 13

Source: Created by the author

Figure 48 – Non-Modularized - Question 3.

```
public class Dates {
    public static void main(String[] args) {
        int val = 90 + 7 + 365 - 30 - 1;
        System.out.print( val );
    }
}
```

1) 431      2) 460      3) 430      4) 461      5) 423

Source: Created by the author

Figure 49 – Non-Modularized - Question 4.

```
public class Queue {
    public static void main(String[] args) {
        int[] tasks = { 3, 6, 4 };
        int total = 0;
        for ( int count = 0; count < tasks.length; count++ ) {
            total = total + ( ( tasks[count] + 2 ) * 2 );
        }
        System.out.print( total );
    }
}
```

1) 40      2) 38      3) 36      4) 30      5) 42

Source: Created by the author

Figure 50 – Non-Modularized - Question 5.

```
public class Cards {
    public static void main(String[] args) {
        int[] tasks = { 8, 6, 3, 9, 0, 1 };
        int amount = 0;
        for ( int counter = 0; counter < tasks.length; counter++){
            switch ( tasks[counter] ) {
                case 1: amount = amount + 3;
                       break;
                case 3: amount = amount + 4;
                       break;
                case 0: amount = amount + 2;
                       break;
                default: amount = amount + 1;
                       break;
            }
        }
        System.out.print( amount );
    }
}
```

1) 10      2) 12      3) 11      4) 13      5) 14

Source: Created by the author





**APPENDIX G – MODULARIZED QUESTIONS**

Figure 51 – Modularized - Question 1.

```
public class Payment {
    public static void main(String[] args) {
        double bill = 13;
        payIfNecessary ( bill - 7 );
    }
    public static void payIfNecessary ( double amount) {
        calculateAndDeliverPay ( amount * 2 );
    }
    private static void calculateAndDeliverPay ( double pay ) {
        deliveryPay ( pay + 15 );
    }
    private static void deliveryPay ( double value ) {
        calculateValue ( value / 3 );
    }
    private static void calculateValue ( double total ) {
        System.out.print( ++total );
    }
}
```

1) 11      2) 13      3) 12      4) 10      5) 9

Source: Created by the author

Figure 52 – Modularized - Question 2.

```
public class Score {
    public int calculatePoints ( int gols ) {
        return ( gols + 1 ) * 3;
    }
}

public class Soccer {
    public static void main(String[] args) {
        Score points = new Score();
        int total = points.calculatePoints ( 3 );
        total = total + points.calculatePoints ( 1 );
        total = total + points.calculatePoints ( 2 );
        System.out.print( total );
    }
}
```

1) 25      2) 24      3) 29      4) 21      5) 27

Source: Created by the author

Figure 53 – Modularized - Question 3.

```
public class Day {
    public int add ( int v ) {
        return v + 1;
    }
}

public class Week {
    public int remove ( int v ) {
        return v - 7;
    }
}

public class Dates {
    public static void main(String[] args) {
        int val = 100;
        val = new Week().remove(val);
        val = new Year().add(val);
        val = new Month().remove(val);
        val = new Day().add(val);
        System.out.print( val );
    }
}

public class Month {
    public int remove ( int v ) {
        return v - 30;
    }
}

public class Year {
    public int add ( int v ) {
        return v + 365;
    }
}
```

1) 452    2) 458    3) 428    4) 459    5) 429

Source: Created by the author

Figure 54 – Modularized - Question 4.

```
public class Time {
    public int delay ( int val ) {
        return ( val + 2 ) * 2;
    }
}

public class Queue {
    public static void main(String[] args) {
        int[] tasks = { 2, 7, 3 };
        int total = 0;
        Time clock = new Time();
        for ( int counter = 0; counter < tasks.length; counter++ ) {
            total = total + clock.delay ( tasks[counter] );
        }
        System.out.print( total );
    }
}
```

1) 40    2) 42    3) 30    4) 36    5) 38

Source: Created by the author

Figure 55 – Modularized - Question 5.

```
public class Cards {
    public static void main(String[] args) {
        int[] tasks = { 4, 2, 7, 6, 3, 2 };
        int qtd = 0;
        for ( int n = 0; n < tasks.length; n++ ) {
            switch ( tasks[n] ) {
                case 2: TypeA a = new TypeA();
                    qtd = a.plus ( qtd );
                    break;
                case 4: TypeB b = new TypeB();
                    qtd = b.plus ( qtd );
                    break;
                case 6: TypeC c = new TypeC();
                    qtd = c.plus ( qtd );
                    break;
                default: TypeD d = new TypeD();
                    qtd = d.plus ( qtd );
                    break;
            }
        }
        System.out.print( qtd );
    }
}

public class TypeA {
    public int plus ( int v1 ) {
        return v1 + 2;
    }
}

public class TypeB {
    public int plus ( int op ) {
        return op + 5;
    }
}

public class TypeC {
    public int plus ( int fy ) {
        return fy + 3;
    }
}

public class TypeD {
    public int plus ( int ts ) {
        return ts + 1;
    }
}

1) 13      2) 10      3) 12      4) 11      5) 14
```

Source: Created by the author



## ANNEX A – FREE AND EXCLUDED CONSENT TERM

### Termo de Consentimento Livre e Esclarecido

Eu, \_\_\_\_\_ (seu nome), portador do RG \_\_\_\_\_, estou sendo convidado a fazer parte de um estudo intitulado de: **técnica para mensurar carga cognitiva na abstração de artefatos de software**, cujos objetivos e justificativas são: **identificar padrões de carga cognitiva em diferentes níveis de complexidade a fim entender e estimar a compreensibilidade em diferentes estruturas**. A minha contribuição para este estudo é a de: **observar, interpretar e responder perguntas de referente a artefatos de software e realizar pequenas tarefas de calibração**.

Fui informado que posso me recusar de participar do estudo, ou até de retirar meu consentimento a qualquer momento durante o estudo, sem precisar justificar ou dar esclarecimentos, e por desejar sair da pesquisa, não irei sofrer qualquer forma de prejuízo à assistência que estou recebendo. Foi esclarecido que eu posso optar por métodos alternativos antes do experimento, que são: **a mudança de qualquer equipamento utilizado durante o experimento**.

Foram me repassados todos os esclarecimentos necessários sobre quaisquer possíveis desconfortos e riscos em relação a este estudo, levando-se em conta que é uma pesquisa, e os resultados positivos ou negativos somente serão obtidos após a sua realização. Assim, **desconfortos devido a utilização do EPOC+ da marca Emotiv podem ser notados após longo tempo de exposição, sem nenhum outro risco envolvido**. Também estou ciente de que minha privacidade será respeitada, ou seja, meu nome ou qualquer outro dado ou elemento que possa, de qualquer forma, me identificar, será mantido de forma confidencial. De qualquer forma, caso qualquer tipo de dano decorrente da minha participação no estudo, conforme determina a lei, eu deverei ser indenizado.

Os pesquisadores que participam deste projeto são: **o mestrando Matheus Segalotto da Universidade do Vale do Rio do Sinos (Unisinos) e o orientador doutor Kleinner Farias da Universidade do Vale do Rio do Sinos (Unisinos)**. Eu poderei manter contato com o mestrando através do telefone **+55 51 98444-2990** ou simplesmente pelo e-mail **matheus.segalotto@gmail.com**.

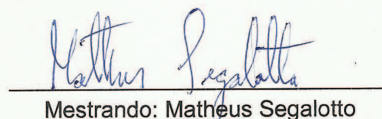
O livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo e suas consequências é assegurada a assistência durante toda a pesquisa, tendo sido também orientado quanto ao teor de todo o aqui mencionado e compreendido a natureza e o objetivo do já referido estudo, exponho meu livre consentimento em fazer parte, estando completamente ciente de que não há nenhum valor econômico, a receber ou a pagar, por minha participação, ou seja, não irá ocorrer nenhum tipo de benefícios, lícito ou ilícito, a não ser os resultados que o trabalho do mestrando venha a atingir no fim deste estudo.

Recebi a informação que em caso de reclamação, abuso, ou denúncia sobre este estudo devo mandar e-mail para **pipca@unisinos.br** ou ligar para o PIPCA **+55 51 93590-8161**.

São Leopoldo, \_\_\_\_\_ de \_\_\_\_\_ de 20\_\_\_\_.



Orientador: Kleinner Farias



Mestrando: Matheus Segalotto

\_\_\_\_\_  
Nome do Participante da Pesquisa

\_\_\_\_\_  
Assinatura do Participante



## ANNEX B – CONSENT LETTER



UNIVERSIDADE DO VALE DO RIO DOS SINOS  
Programa de Pós-Graduação em Computação Aplicada – PIPCA

**Carta de Anuência**

Eu, Sandro José Rigo, coordenador do programa de mestrado em computação aplicada na Unisinos, aceito o pesquisador principal Matheus Segalotto, sob responsabilidade e orientação do Prof. Dr. Kleinner Farias no resguardo da segurança e bem-estar dos sujeitos de pesquisa nela recrutados, dispondo de infraestrutura necessária para a garantia de tal segurança e bem-estar.

Esta instituição, Universidade do Vale do Rio dos Sinos, está ciente de suas co-responsabilidades como instituição co-participante do projeto de pesquisa intitulado Técnica para mensurar carga cognitiva na abstração de artefatos de software, cujos objetivos e justificativas são: identificar padrões de carga cognitiva em diferentes níveis de complexidade a fim entender e estimar a compreensibilidade em diferentes estruturas.

Ciente dos objetivos e da metodologia da pesquisa acima citada, concedo a anuência para seu desenvolvimento, desde que me sejam assegurados os requisitos abaixo:

- O cumprimento das determinações éticas da Resolução nº466/2012 CNS/CONEP e a garantia de solicitar e receber esclarecimentos antes, durante e depois do desenvolvimento da pesquisa.
- Não haverá nenhuma despesa para esta instituição que seja decorrente da participação dessa pesquisa, e no caso do não cumprimento dos itens acima, a liberdade de retirar minha anuência a qualquer momento da pesquisa sem penalização alguma.

São Leopoldo, 10/3/2017

**UNISINOS**

  
**Prof. Dr. SANDRO JOSÉ RIGO**  
Coordenador do Programa Interdisciplinar  
de Pós-Graduação em Computação Aplicada

Assinatura e carimbo do responsável institucional

  
Local e Data da assinatura