Programa de Pós-Graduação em

Computação Aplicada

Doutorado Acadêmico

Lucian José Gonçales

CogEff: Uma abordagem para mensurar a carga cognitiva de desenvolvedores em tarefas de compreensão de código

São Leopoldo, 2022

Lucian José Gonçales

COGEFF:
Uma abordagem para mensurar a carga cognitiva de desenvolvedores em tarefas de compreensão de código

Tese apresentada como requisito parcial para a obtenção do título de Doutor pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Kleinner Silva Farias de Oliveira

São Leopoldo
2022

G635c Gonçales, Lucian José.
CogEff : uma abordagem para mensurar a carga cognitiva de desenvolvedores em tarefas de compreensão de código / Lucian José Gonçales. – 2022.
217 f. : il. ; 30 cm.

Tese (doutorado) – Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2022.
"Orientador: Prof. Dr. Keinner Silva Farias de Oliveira."

1. Carga cognitiva. 2. Compreensão de código. 3. Engenharia de software. I. Título.

CDU 004

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecária: Silvana Dornelles Studzinski – CRB 10/2524)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) - Código de Financiamento 001 / *This study was financed in partby the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.*

Lucian José Gonçales

CogEff:
UMA ABORDAGEM PARA MENSURAR A CARGA COGNITIVA DE DESENVOLVEDORES EM TAREFAS DE COMPREENSÃO DE CÓDIGO

Tese apresentada à Universidade do Vale do Rio dos Sinos – UNISINOS, como requisito parcial para obtenção do título de Doutor em Computação Aplicada pelo Programa de Pós-Graduação em Computação Aplicada – PPGCA.

Aprovado em 31 Março 2022

BANCA EXAMINADORA

Prof. Dr. Bruno Carreiro da Silva – California Polytechnic State University (Cal Poly)

Prof. Dr. Everton Tavares Guimarães – Pennsylvania State University (Penn State)

Prof. Dr. Jorge Luis Victória Barbosa – Universidade do Vale do Rio dos Sinos (UNISINOS)

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

Visto e permitida a impressão
São Leopoldo,

Prof. Dr. Rafael Kunst
Coordenador PPG em Computação Aplicada

Eu dedico esse estudo em especial à minha mãe e à meu pai.
Obrigado a vocês pelo suporte e incentivo.

A breeze will always blow in the
direction that it wishes to go.
— ANTHONY T. HINCKS

AGRADECIMENTOS

Gostaria de agradecer a Deus em primeiro lugar. Sem a vida que ele me deu, nada disso seria possível. Em seguida, várias pessoas me ajudaram e prestaram suporte à condução desse trabalho. Em especial, agradeço ao meu orientador Prof. Dr. Kleinner Silva Farias de Oliveira. Sem ele, eu não teria conseguido ingressar no doutorado. Ele prestou orientação na elaboração da proposta para ingressar no programa de pós-graduação.

Também me ajudou na consolidação do tema. O Dr. Kleinner é um pesquisador que aposta no potencial do uso de técnicas da neurociência para melhorar práticas e artefatos na engenharia de software. Além disso, prestou suporte na revisão dos artigos relacionados a esta tese. Também me orientou em como a técnica da CogEff poderia ser implementada, bem como avaliada. Muito obrigado Kleinner, por sua dedicação a este trabalho.

Agradeço também ao Prof. Dr. Bruno Carreiro da Silva, Prof. Dr. Everton Tavares Guimarães, e Prof. Dr. Jorge Luis Victória Barbosa pela participação da banca avaliadora desta tese, e por seus respectivos valiosos retornos que foram cuidadosamente implementados neste trabalho após a avaliação da defesa desta tese.

Gostaria de agradecer aos colegas e ex-colegas do Programa de Pós-Graduação em Computação Aplicada (PPGCA) que colaboraram com a evolução deste trabalho. Também agradeço em geral a todos os professores do Programa de Pós-Graduação em Computação Aplicada (PPGCA).

Agradeço a agência de fomento CAPES. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001. Sem o apoio da CAPES, jamais teria condições de adquirir os materiais para conduzir esta pesquisa, e permanecer dedicado integralmente à pesquisa dentro da universidade.

Agradeço também a Universidade do Vale do Rio dos Sinos (UNISINOS) por prover todo o espaço, infraestrutura para execução da pesquisa e a execução dos experimentos.

Por fim, a contribuição de vocês foi muito importante para a evolução e andamento deste trabalho! Muito Obrigado!

"In the past, many attempts have been made to use software measures, such as code complexity, to understand why certain programming constructs or idioms may be more difficult to understand. Recent studies have tried to predict comprehensibility of source code based on a combination of software measures, but could only show small predictive power. All these approaches suffer from two main limitations: (1) software measures lack justification for cognitive outcomes; and as a result, current software measures have poor predictive power on program comprehension, and (2) software measures are not explanatory, and cannot provide an answer to why certain constructs are more difficult than others... Can we create cognition-based software measures to overcome these limitations?"

(Norman Peitek)

RESUMO

A carga cognitiva refere-se ao esforço mental que os usuários aplicam ao desempenhar tarefas cognitivas tais como interpretar artefatos de software com diferentes níveis de abstração. A carga cognitiva na engenharia de software é importante devido ao potencial de considerar fatores humanos através de sinais fisiológicos enquanto desenvolvedores estão em sua rotina de trabalho. Por exemplo, enquanto estão trabalhando, desenvolvedores dedicam grande parte de seu tempo na compreensão de código. Mensurar a carga cognitiva através de indicadores psicofisiológicos possibilitaria uma correlação coerente da percepção dos desenvolvedores com as tarefas de compreensão de código. Enquanto indicadores psicofisiológicos refletem instantaneamente os estímulos dos desenvolvedores, métricas tradicionais apenas são consolidadas após a conclusão das tarefas. Por isso, abordagens de carga cognitiva são apontadas por ter potencial preditivo nas tarefas de engenharia de software. Para investigar a carga cognitiva na engenharia de software, pesquisadores combinam abordagens a partir de variados tipos de dispositivos psicofisiológicos. Por exemplo, a partir do eletroencefalograma (EEG) pesquisadores já utilizaram diretamente abordagens tais como o *Assymetry Ratio* (ASR), o *Event-Related Desyncronization* (ERD), e as potências de banda das ondas alfa (α), beta (β), delta (δ), e theta (θ). A partir do sensor fMRI, pesquisadores também utilizaram o *Blood Oxygen Level Dependent* (BOLD) para realçar quais áreas cerebrais são ativas durante a compreensão de código. Porém, apesar dessas técnicas serem relacionadas à carga cognitiva, ainda falta uma abordagem para mensurar a carga cognitiva em tarefas de compreensão de código. Isto implica em uma série de problemas identificados nesta área: (1) ausência de uma classificação do estado da arte sobre medidas da carga cognitiva na engenharia de software; (2) falta de abordagem para mensurar a carga cognitiva na engenharia de software; (3) escassez de análises de correlação de abordagens utilizadas para mensurar a carga cognitivas em tarefas de compreensão de código; (4) ausência da avaliação da efetividade para a utilização de dados EEG para classificar a compreensão de código utilizando técnicas de *machine learning*. Por isso, esta pesquisa busca: (1) realizar um estudo de mapeamento sistemático para classificar as pesquisas sobre medidas da carga cognitiva na engenharia de software; (2) desenvolver uma técnica para mensurar a carga cognitiva através de dados EEG em tarefas de compreensão de software, chamada de CogEff; (3) análise da correlação entre abordagens de EEG tradicionais, e da abordagem CogEff em relação a tarefas de compreensão de código; (4) análise da efetividade ao utilizar abordagens tradicionais de EEG para classificar compreensão de código. Os principais resultados são: (1) a partir da classificação dos estudos foi constatado que 37% (23/63) dos estudos adotaram dispositivos multimodais; e 59% (37/63) dos estudos analisaram a carga cognitiva em tarefas de programação, tais como, compreensão de código; (2) abordagem CogEff possui potencial de medir a carga cognitiva através da conectividade dos canais EEG; (3) os testes de correlação apresentaram que as abordagens de EEG tradicionais, e a CogEff possuem correlação com a compreensão de código; (4) o classificador *K-Nearest Neighbors* obteve uma média de *f-measure* de 86% para classificar a compreensão do código.

Palavras-chave: Compreensão de código. Carga cognitiva. Engenharia de Software.

ABSTRACT

Cognitive load refers to the mental effort that users spend when performing cognitive tasks such as interpreting software artifacts with different levels of abstraction. Cognitive load in software engineering is important due to the potential for considering human factors through physiological signals while users are in their work routine. For instance, developers spend most of their time understanding code while working. Measuring the cognitive load using psychophysiological indicators would enable a coherent correlation between the developers' perception and the code comprehension tasks. While psychophysiological indicators instantly reflect user stimuli, traditional metrics consolidate after comprehension tasks finish. Therefore, literature recognizes that cognitive load approaches have high predictive potential in software engineering tasks. Researchers have investigated the cognitive load in software engineering by combining various approaches from different psychophysiological devices. For example, from the electroencephalogram (EEG) researchers have already directly used approaches such as as the Assymetry Ratio (ASR), the Event-Related Desyncronization (ERD), and the band power of the alpha (α), beta (β), delta (δ), and theta (θ) waves. Using the fMRI sensor, researchers also used the *Blood Oxygen Level Dependent* (BOLD) to highlight which brain areas are active during code comprehension tasks. However, despite these techniques' relation to cognitive load, an approach to measure cognitive load in code comprehension tasks is still lacking. Thus, there is a series of problems identified in this area: (1) Absence of a state-of-the-art classification on measures of cognitive load in software engineering; (2) Lack of approach to measure cognitive load in software engineering; (3) Absence of correlation analysis of approaches used to measure cognitive load in code comprehension tasks; (4) Lack of evaluation of effectiveness for using EEG data to classify code comprehension using *machine learning* techniques. Therefore, this research aims to: (1) Conduct a systematic mapping study to classify research on measures of cognitive load in software engineering; (2) Develop a technique to measure cognitive load through EEG data in software comprehension tasks, named as CogEff; (3) Analyze the correlation between traditional EEG approaches, and the CogEff approach with code comprehension tasks; (4) Analyze the effectiveness of classifying code comprehension trained with traditional EEG approaches. The main results are: (1) Based on the classification, 37% (23/63) of the studies adopted multimodal devices; and 59% (37/63) of the studies analyzed the cognitive load in programming tasks, such as code comprehension; (2) CogEff approach has the potential to measure cognitive load through EEG channel connectivity; (3) Correlation tests showed that traditional EEG approaches, and CogEff correlates with code comprehension; (4) The K-Nearest Neighbors classifier obtained an average f-measure of 86% to classify code comprehension based on EEG data.

Keywords: Code comprehension. Cognitive load. Software engineering.

LISTA DE FIGURAS

Figura 1 – Compreensibilidade dos desenvolvedores em relação ao código fonte. . . . 34
Figura 2 – Esquema da problemática investigada. 35
Figura 3 – Áreas de Broadmann do cérebro que são ativadas durante a compreensão do código. 46
Figura 4 – Processo de medida da carga cognitiva na engenharia de software. 47
Figura 5 – Resultados de cada etapa do processo de filtragem. 57
Figura 6 – Os resultados do processo de *snowballing* em relação à cada etapa. 58
Figura 7 – A distribuição dos participantes em relação às tarefas. 82
Figura 8 – Histórico de publicações dos estudos primários selecionados. 85
Figura 9 – Esquema de classificação da carga cognitiva na Engenharia de Software. . . 87
Figura 10 – Gráfico de bolhas que relaciona três variáveis. O *eixo-x* consiste no método de pesquisa utilizado (A) e nas contribuições (B). O *eixo-y* representa o propósito dos estudos primários. O tamanho das bolhas consiste na quantidade de estudos primários classificados de acordo com os critérios do *eixo-x* e *eixo-y*. 90
Figura 11 – Visão geral do processo proposto para mensurar carga cognitiva. 110
Figura 12 – Exemplos de conexões com pesos de acordo com as propriedades definidas. 117
Figura 13 – Canais do Emotiv EPOC de acordo com o padrão internacional 10-20. . . . 119
Figura 14 – Exemplo de tarefa de compreensão de código geralmente empregada nos estudos experimentais. 121
Figura 15 – Exemplo em execução da abordagem CogEff. 126
Figura 16 – Visão geral do fluxo de processamento dos dados EEG. 138
Figura 17 – Representação da distribuição dos dados. 142
Figura 18 – Representação da distribuição dos dados da Hipótese 3. 143
Figura 19 – Representação da correlação de *kendall* entre abordagens EEG e respostas de todas as tarefas de compreensão de código. 146
Figura 20 – Representação da correlação de *Pearson* entre as abordagens EEG e o tempo para concluir as tarefas de compreensão. 149
Figura 21 – Visão geral do fluxo de processamento dos dados EEG. 158
Figura 22 – Distribuição da precisão, *recall*, e *f-measure* das técnicas de *machine learning*. 162
Figura 23 – Resultados da precisão das técnicas de *machine learning*. 163
Figura 24 – Resultados do *recall* das técnicas de *machine learning*. 164
Figura 25 – Resultados da *f-measure* das técnicas de *machine learning*. 165
Figura 26 – Resultados do MCC das técncias de *machine learning*. 166
Figura 27 – Representação da efetividade das MLT treinadas com a CogEff. 170
Figura 28 – Representação da distribuição da efetividade das MLT treinadas com as (A) Abordagens tradicionais de EEG e (B) Abordagens tradicionais de EEG e CogEff. 171
Figura 29 – Tarefa T1 - Processar Salário. 203
Figura 30 – Tarefa T2 - Processar Salário v2. 203
Figura 31 – Tarefa T3 - Processar Pontos. 204
Figura 32 – Tarefa T4 - Processar Pontos v2. 204
Figura 33 – Tarefa T5 - Obter Calendário. 204
Figura 34 – Tarefa T6 - Obter Calendário v2. 205
Figura 35 – Tarefa T7 - Contador Elementos Fila. 205
Figura 36 – Tarefa T8 - Contador Elementos Fila v2. 206

Figura 37 – Tarefa T9 - Contador de Cartões. 206
Figura 38 – Tarefa T10 - Contador de Cartões v2. 207

LISTA DE TABELAS

Tabela 1 – Questões de pesquisa definidas 52
Tabela 2 – Definição das principais palavras chave e seus sinônimos que compõem a *string* de busca. 54
Tabela 3 – Lista dos motores de busca selecionados 55
Tabela 4 – Dados que foram extraídos dos estudos selecionados. 59
Tabela 5 – Classificação das definições de carga cognitiva utilizadas nos estudos primários. 61
Tabela 6 – Classificação dos tipos de carga cognitiva a partir da Teoria da Carga Cognitiva. 62
Tabela 7 – Classificação de sensores para medir a carga cognitiva dos desenvolvedores. 63
Tabela 8 – Classificação dos estudos primários que utilizaram múltiplos sensores. . . . 65
Tabela 9 – Classificação de estudos primários baseados nas métricas relacionadas à carga cognitiva. 66
Tabela 10 – Classificação de estudos primários baseados em métricas multimodais. . . . 68
Tabela 11 – Técnicas de *machine learning* utilizadas com métricas da carga cognitiva. . 69
Tabela 12 – Lista de estudos primários que aplicaram mais de uma técnica de *machine learning*. 70
Tabela 13 – Lista dos conjuntos de dados produzidos pelos estudos primários. 72
Tabela 14 – Classificação de estudos primários baseados no propósito para medir a carga cognitiva. 74
Tabela 15 – Classificação de estudos primários em relação às tarefas de Engenharia de Software. 76
Tabela 16 – Classificação de estudos primários de acordo com os tipos de tarefas de programação. 78
Tabela 17 – Classificação dos estudos primários em relação aos artefatos. 78
Tabela 18 – Lista de linguagens de programação utilizadas nos estudos primários. . . . 80
Tabela 19 – Classificação dos estudos primários baseados na quantidade de participantes. 81
Tabela 20 – Classificação de estudos primários baseados no método de pesquisa. 83
Tabela 21 – Lista de estudos primários classificados como pesquisa de avaliação. 84
Tabela 22 – Principais características dos trabalhos relacionados. 98
Tabela 23 – Lacunas dos trabalhos relacionados. 106
Tabela 24 – Legenda da notação contida na Figura 12. 116
Tabela 25 – Análise comparativa entre as abordagens tradicionais utilizadas pela literatura de engenharia de software para mensurar a carga cognitiva. 130
Tabela 26 – Descrição das tarefas de compreensão. 139
Tabela 27 – Estatística descritiva das variáveis psicofisiológicas. 141
Tabela 28 – Estatística descritiva da CogEff nas tarefas de diferentes níveis de complexidade. 142
Tabela 29 – Resultado da correlação de *Kendall* na hipótese 1 144
Tabela 30 – Resultado dos testes de correlação de *Spearman* para a hipótese 1. 145
Tabela 31 – Resultado dos testes da Hipótese 2 com correlação de *Pearson*. 148
Tabela 32 – Resultado dos testes da Hipótese 2 com correlação de *Spearman*. 148
Tabela 33 – Resultado da Hipótese 3 com o teste de Wilcoxon 150
Tabela 34 – Resultado da Hipótese 3 com o *t-test*. 151
Tabela 35 – Categoria das métricas de efetividade. 160

Tabela 36 – Resultado da precisão, *recall* e *f-measure* das respectivas técnicas de *machine learning*. 163
Tabela 37 – Resultados do *t-test* de uma amostragem das *f-measures* dos classificadores, valor testado igual a 0.8. 167
Tabela 38 – Resultado do *t-test* de uma amostragem da *f-measure* dos classificadores. . 168
Tabela 39 – Efetividade (*f-measure*) das técnicas de *machine learning* treinada com as abordagens EEG tradicionais e CogEff. 170
Tabela 40 – Resultado do *t-test* entre a efetividade (*f-measure*) das MLTs treinadas com as abordagens tradicionais de EEG e das MLTs treinadas com as abordagens tradicionais de EEG e a CogEff. 172
Tabela 41 – Resultado do teste de Wilcoxon entre a efetividade (*f-measure*) das MLTs treinadas com as abordagens tradicionais de EEG, e das MLTs treinadas com as abordagens tradicionais de EEG e a CogEff. 173
Tabela 42 – Classificador *K-Nearest Neighbor*: resultados da precisão, *recall* e *f-measure*. 215
Tabela 43 – Classificador *Neural Network*: resultados da precisão, *recall* e *f-measure*. . 215
Tabela 44 – Classificador *Naïve Bayes*: resultados da precisão, *recall* e *f-measure*. . . . 215
Tabela 45 – Classificador *Random Forest*: resultados da precisão, *recall* e *f-measure*. . . 216
Tabela 46 – Classificador *Support Vector Machine*: resultados da precisão, *recall* e *f-measure*. 216
Tabela 47 – Classificador aleatório/*Naive*: resultados da precisão, *recall* e *f-measure*. . . 216
Tabela 48 – Classificador *K-Nearest Neighbor* treinado com dados psicofisiológicos e os resultados da CogEff: resultados da precisão, *recall* e *f-measure*. 217
Tabela 49 – Classificador *Neural Network* treinado com dados psicofisiológicos e da CogEff: resultados da precisão, *recall* e *f-measure*. 217
Tabela 50 – Classificador *Random Forest* treinado com dados psicofisiológicos e da CogEff: resultados da precisão, *recall* e *f-measure*. 217

LISTA DE ALGORITMOS

Algoritmo 1 – Algoritmo para filtragem do sinal EEG 122
Algoritmo 2 – Montagem de grafo e estabelecimento de conexões 124
Algoritmo 3 – Densidade do grafo 125

LISTA DE EQUAÇÕES

Equação 4.1 – Soma da convolução executada pelo filtro FIR. 111
Equação 4.2 – Janela de Hamming. 112
Equação 4.3 – Modelo generativo da Análise de Componentes Independentes. . . . 112
Equação 4.4 – Subtração de vetores utilizada no algoritmo *fast* ICA. 113
Equação 4.5 – Normalização da subtração de vetores. 113
Equação 4.6 – Definição da transformada rápida de Fourier. 113
Equação 4.7 – Cálculo do índice de conectividade. 114
Equação 4.8 – Cálculo da densidade do grafo. 118
Equação 6.1 – Precisão 160
Equação 6.2 – *Recall* 160
Equação 6.3 – *F-measure* 160

LISTA DE ABREVIATURAS

CogEff	Cognitive Effort
Conf.	Conference
Comp.	Computing
Eng.	Engineering
Hum.	Human
Int.	International
Interact.	Interaction
e.g.	Exempli gratia - Por exemplo
et al.	et alia - e outros
i.e.	id est - isto é
Lec.	Lecture
pp.	Páginas
Proc.	Proceedings
s.l.	Sine loco - Sem local.
s.n.	Sine nomine - Sem nome.
Soft.	Software
Symp.	Symposium
Tech.	Technology
n.	Número
v.	Volume

LISTA DE SIGLAS

BA	Broadmann Area
BCI	Brain-Computer Interface
BPMN	Business Process Model and Notation
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CE	Critérios de Exclusão
CI	Critério de Inclusão
ECG	Eletrocardiograma
EDA	Electro-Dermal Activity
EEG	Eletroencefalograma
ERD	Event Related Desynchronization
fMRI	Functional Magnetic Resonance Imaging
fNIRS	Functional near-infrared spectroscopy
GSR	Galvanic Skin Response
HbO	Hemoglobin Deoxygenated
HbR	Hemoglobin Oxygenated
HbT	Hemoglobin Total
HCI	Human Computing Interaction
HR	Heart Rate Variability
HRV	Heart Rate Variability
IAF	Individual Alpha Frequency
KNN	K-Nearest Neighbors
MEG	Magnetoencefalografia
MT	Metodologia
NB	Naïve Bayes
NB	Neural Network
PLV	Phase Locking Value
Oxi	Oxigenação
PIO	Population, Intervention, Outcome
PCA	Principal Component Analysis
PS	Potência do Sinal
QP	Questão de pesquisa
RF	Random Forrest
RR	Respiratory Rate

SCL	Skin Conductance Level
SS	Search String
SMS	Systematic Mapping Study
SVM	Support Vector Machine
UML	Unified Modeling Language
VOI	Volume of Interest

LISTA DE SÍMBOLOS

α	Alfa
β	Beta
γ	Gamma
ε	Epsilon
θ	Theta
Δ	Delta
ν	Ni
eff	Esforço
$\rightarrow$	Implica
E_n	Quantidade de arestas ausentes.
C_n	Grafo de conectividade de Compreensão de tamanho n nodos.
H_n	Grafo de conectividade de Hemisfério de tamanho n nodos.
R_n	Grafo de conectividade de Regiões de tamanho n nodos.
$>$	Maior que
$\leq$	Menor igual
tp	True positive
tn	True negative
fn	False negative
fp	False positive

SUMÁRIO

1 INTRODUÇÃO 33
1.1 Motivação e Problemática . . . 33
1.2 Objetivos e Questões de Pesquisa . . . 38
1.3 Metodologia . . . 41
1.4 Contribuições . . . 43
1.5 Estrutura . . . 44

2 FUNDAMENTAÇÃO TEÓRICA 45
2.1 Carga cognitiva . . . 45
2.2 Implicações da carga cognitiva nos desenvolvedores . . . 45
2.3 Processo de mensuramento da carga cognitiva na Engenharia de Software . 47
2.4 Eletroencefalografia . . . 48
2.5 Considerações finais . . . 49

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA 51
3.1 Metodologia . . . 51
3.1.1 Objetivos de pesquisa . . . 52
3.1.2 Estratégia de busca . . . 53
3.1.3 Critérios de inclusão e exclusão de estudos . . . 55
3.1.4 Processos de seleção de estudos primários . . . 56
3.1.5 Extração de dados . . . 59
3.1.6 Síntese de dados . . . 59
3.2 Resultados . . . 60
3.2.1 QP1: como a carga cognitiva foi definida nos estudos? . . . 61
3.2.2 QP2: quais são os tipos de sensores adotados para capturar dados relacionados a carga cognitiva dos desenvolvedores? . . . 63
3.2.3 QP3: quais métricas foram usadas para medir a carga cognitiva dos desenvolvedores? . . . 66
3.2.4 QP4: quais algoritmos foram usados para classificar a carga cognitiva do desenvolvedor? . . . 69
3.2.5 QP5: para qual propósito a carga cognitiva dos desenvolvedores foi medida? . . . 73
3.2.6 QP6: quais tarefas foram usadas para medir a carga cognitiva dos desenvolvedores? 75
3.2.7 QP7: quais foram os artefatos usados nas tarefas cognitivas? . . . 78
3.2.8 QP8: quantos participantes os estudos recrutaram para medir a carga cognitiva? . 80
3.2.9 QP9: quais métodos de pesquisa foram utilizados para investigar a carga cognitiva nas tarefas de desenvolvimento de software? . . . 83
3.2.10 QP10: onde os estudos foram publicados? . . . 85
3.2.11 Esquema de classificação . . . 86
3.3 Discussão dos resultados . . . 89
3.3.1 Gráfico de bolhas . . . 89
3.3.2 Contribuições . . . 91
3.3.3 Oportunidades e desafios futuros de pesquisa . . . 93
3.4 Trabalhos relacionados ao estudo sistemático . . . 96
3.5 Ameaças a validade . . . 99
3.6 Lista de trabalhos relacionados à abordagem CogEff . . . 100
3.7 Análise Comparativa . . . 105

3.8 Considerações finais . . . 107

4 COGEFF: UMA ABORDAGEM PARA MENSURAR A CARGA COGNITIVA EM TAREFAS DE COMPREENSÃO DE CÓDIGO 109
4.1 Visão Geral da CogEff . . . 109
4.1.1 Etapa 1: Filtrar sinais EEG . . . 111
4.1.2 Etapa 2: Montagem do grafo e definições de propriedades . . . 113
4.1.3 Etapa 3: Densidade do grafo . . . 118
4.2 Instrumentação . . . 119
4.3 Tarefas . . . 120
4.4 Escala da carga cognitiva da CogEff . . . 121
4.5 Implementação . . . 122
4.5.1 Filtragem EEG . . . 122
4.5.2 Montagem do grafo e atribuição de pesos . . . 123
4.5.3 Densidade do grafo . . . 124
4.6 Demonstração da Execução da Abordagem CogEff . . . 125
4.7 Análise comparativa das abordagens correlacionadas . . . 127
4.7.1 Critérios da análise comparativa . . . 129
4.8 Considerações finais . . . 130

5 AVALIAÇÃO DA COGEFF 133
5.1 Objetivo e questões de pesquisa . . . 133
5.2 Hipóteses . . . 134
5.3 Perfil dos participantes . . . 136
5.4 Processo experimental . . . 137
5.5 Especificação das tarefas de compreensão . . . 139
5.6 Variáveis . . . 139
5.7 Procedimentos de análise . . . 140
5.8 Resultados . . . 141
5.8.1 Estatística descritiva . . . 141
5.8.2 QP1: Resultados da Hipótese 1 - correlação das abordagens com as respostas das tarefas de compreensão de código . . . 142
5.8.3 QP2: Resultados da Hipótese 2 - correlação das abordagens com o tempo decorrido para resolver tarefas de compreensão de código . . . 147
5.8.4 QP3: Resultados da Hipótese 3 - diferença entre os valores da CogEff entre tarefas de complexidade menor e maior . . . 150
5.9 Ameaças a validade . . . 151
5.10 Considerações finais . . . 152

6 CLASSIFICAÇÃO DE COMPREENSÃO DE CÓDIGO BASEADA EM DADOS DE EEG 155
6.1 Metodologia . . . 156
6.1.1 Objetivo e questões de pesquisa . . . 156
6.1.2 Método para coleta e processamento de dados . . . 157
6.1.3 Métricas de efetividade . . . 159
6.1.4 Procedimentos de análise . . . 161
6.2 Resultados . . . 161
6.2.1 Análise descritiva . . . 161
6.2.2 QP1: Efetividade das técnicas de machine learning . . . 162

6.2.3 QP2: Quais técnicas de aprendizagem de máquina alcançam uma efetividade maior que um classificador aleatório? 167
6.2.4 QP3: Qual é a efetividade dos classificadores de *machine learning* ao adicionar a CogEff no conjunto de dados? 169
6.3 Discussão 173
6.3.1 Influência do tipo de dados nos resultados obtidos pelas técnicas de *machine learning* 173
6.3.2 Implicações 175
6.3.3 Desafios futuros 178
6.4 Ameaças a Validade 181
6.5 Considerações finais 183

7 CONCLUSÃO **185**
7.1 Sumário 185
7.2 Contribuições 188
7.3 Trabalhos Futuros 190

REFERÊNCIAS **193**

ANEXO A – TAREFAS DE COMPREENSÃO **203**

APÊNDICE A – LISTA DE ARTIGOS SUBMETIDOS/PUBLICADOS EM RELAÇÃO A TESE **209**

APÊNDICE B – LISTA DE ESTUDOS PRIMÁRIOS **211**

APÊNDICE C – PRECISÃO, RECALL E F-MEASURE POR CLASSE **215**

1 INTRODUÇÃO

A carga cognitiva refere-se ao esforço mental que os desenvolvedores aplicam ao realizar tarefas que exigem a resolução de problemas em geral (MÜLLER, 2015; LEE et al., 2017). Por exemplo, durante as tarefas de desenvolvimento de software, desenvolvedores gastam carga cognitiva para compreender código antes de tomar decisões e realizar modificações no software. A compreensão de programas consiste no processo cognitivo do desenvolvedor em deduzir o resultado e o propósito de um fragmento de código a partir da observação desse código (SIEGMUND et al., 2017). A carga cognitiva é relevante devido ao seu potencial de incluir e considerar o fator humano durante tarefas de desenvolvimento de software (MÜLLER, 2015; MÜLLER; FRITZ, 2016). Pois, diferentemente das abordagens de medidas de aspectos estáticos do código fonte, tais como o número de linhas, as medidas da carga cognitiva podem ser coletadas a partir dos desenvolvedores antes das tarefas de desenvolvimento de software forem concluídas. As medidas de aspectos estáticos do código fonte são apenas consolidadas após o usuário concluir a tarefa de programação (CRK; KLUTHE, 2016). Além disso, a carga cognitiva é importante porque varia de acordo com o esforço aplicado de cada usuário.

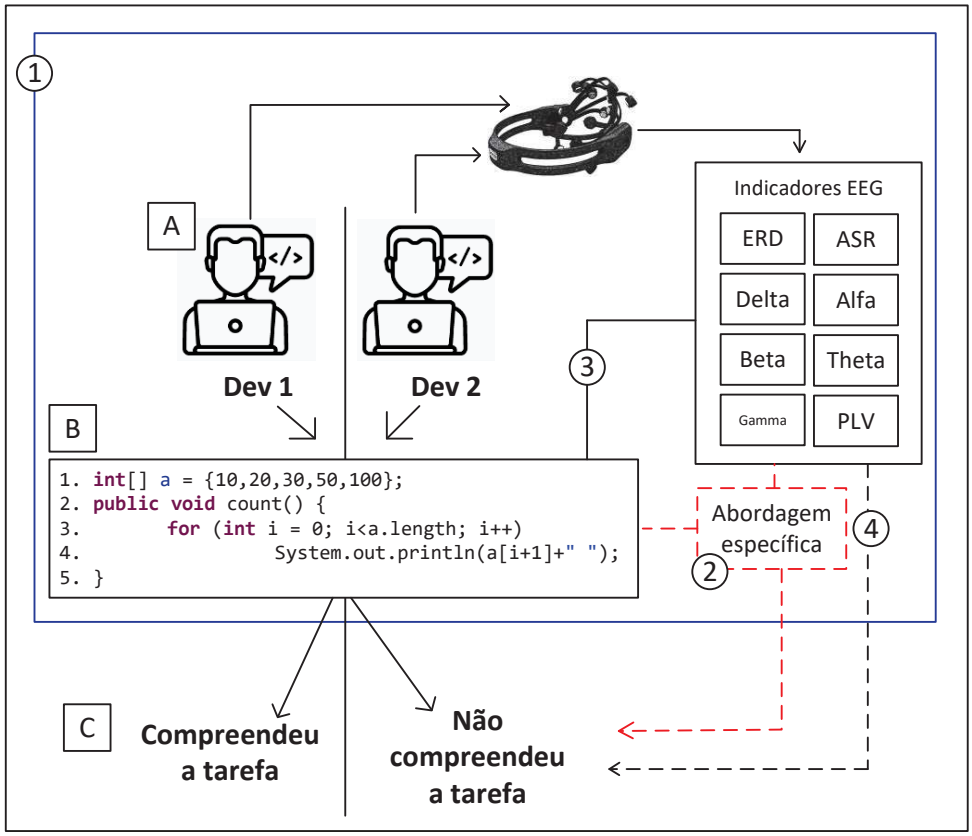
É por isso que pesquisadores na área de Engenharia de Software têm investigado a carga cognitiva. Nessas pesquisas, a carga cognitiva foi utilizada de várias formas, como por exemplo, FRITZ et al. (2014); FRITZ; MÜLLER (2016) utilizaram sensores de EEG, batimentos cardíacos, e temperatura corporal para medir a carga cognitiva dos desenvolvedores de software e classificar o nível de dificuldade dos desenvolvedores em tarefas de programação. (CRK; KLUTHE, 2016; CRK; KLUTHE; STEFIK, 2015) utilizaram o dispositivo de EEG de onde extraíram ondas alfa (α), e theta (θ) para correlacionar com o nível de experiência em programação dos participantes. KOSTI et al. (2018) estabeleceram a conectividade entre os diferentes canais da eletroencefalograma, abordagem conhecida como *Phaze Locking Value* (PLV), para correlacionar o nível de dificuldade dos desenvolvedores em tarefas de programação. LEE et al. (2017) utilizaram a *Assymetric Ratio* (ASR), e o *band power* de várias frequências de sinais tais como alpha (α), beta (β), theta (θ), delta (δ), e gamma (γ) com técnicas de *machine learning* para correlacionar com o nível de experiência e nível de dificuldade dos desenvolvedores de software.

1.1 Motivação e Problemática

Esses indicadores estão sendo utilizados na engenharia de software devido ao potencial de serem coerentes com as percepções individuais dos desenvolvedores. Por exemplo, para indicar nível de conhecimento (CRK; KLUTHE, 2016; CRK; KLUTHE; STEFIK, 2015), e dificuldade em tarefas de programação (FRITZ et al., 2014). A Figura 1 apresenta um cenário onde as métricas baseadas em EEG foram utilizadas em um contexto da engenharia de software. Neste cenário, desenvolvedores (Figura 1.(A)) interagem e resolvem tarefas de engenharia de

software, tais como tarefas de compreensão de código (Figura 1.(B)), onde a compreensão dos artefatos contidos na Figura 1.(B), pode variar entre os desenvolvedores. O resultado dessa interpretação está ilustrado na Figura 1.(C). A motivação para utilização dessas abordagens para mensurar a carga cognitiva na área de engenharia de software é a possibilidade de considerar o fator humano nas atividades de engenharia de software, ao invés de medir aspectos estáticos do código fonte (Figura 1.(B)) para investigar a relação com a compreensão de código dos desenvolvedores. Por exemplo, os estudos empíricos de SCALABRINO et al. (2019, 2017) e TROCKMAN et al. (2018) apontaram que métricas tais como, a quantidade de linhas, e quantidade de métodos do código fonte, não possuem, ou quase não possuem correlação com a compreensão de código pelos desenvolvedores. Desde então, a solução foi adotar abordagens para medir a carga cognitiva dos desenvolvedores, tais como a potência das frequências de banda, *Asymmetry Ratio*, e *Event Related Desyncronization* (ERD) para investigar essa relação a partir da perspectiva dos desenvolvedores (Figura 1.(A)).

Figura 1: Compreensibilidade dos desenvolvedores em relação ao código fonte.



Fonte: Elaborado pelo autor.

Apesar da utilização de várias abordagens de EEG para medir a carga cognitiva dos desenvolvedores, há a falta de uma investigação empírica sobre a relação dessas abordagens com a carga cognitiva de desenvolvedores durante tarefas de compreensão de código. Esta vasta utilização pode ser consequência da falta de senso comum ao adotar as técnicas para medir a carga cognitiva em tarefas de compreensão de código. Para isso, esta tese busca investigar a relação

das abordagens EEG para mensurar carga cognitiva de desenvolvedores. A Figura 1 enumera os pontos de investigação desta tese. Deste modo, este trabalho busca investigar a relação entre as abordagens EEG para medir a carga cognitiva na engenharia de software com tarefas de compreensão de código. Por isso, em relação à Figura 1, a motivação é entender:

1. O estado da arte das medidas da carga cognitiva de desenvolvedores na engenharia de software considerando aspectos, tais como métricas e sensores utilizados, a qual é representado na Figura 1.1;

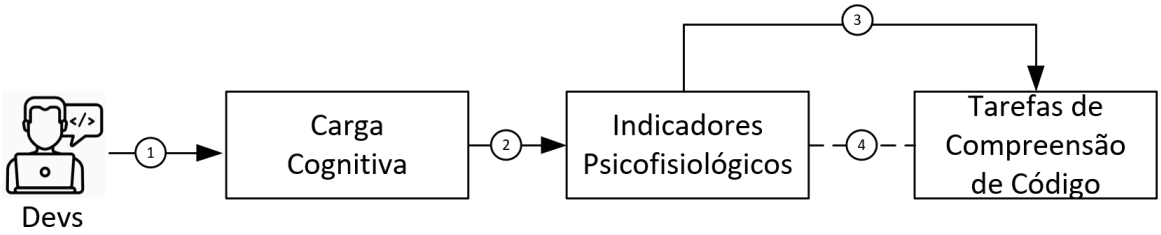
2. Como desenvolver uma abordagem para mensurar carga cognitiva dos desenvolvedores em tarefas de compreensão de código. Esta motivação é representada na Figura 1.2, com as linhas pontilhadas em vermelho indicando uma possível correlação da abordagem, que foi moldada a aspectos das tarefas de compreensão, com as tarefas de compreensão de código;

3. Como essas abordagens se relacionam com algum propósito específico relacionado ao domínio da engenharia de software, tais como a compreensão de código, representado na Figura 1.3; e

4. A efetividade de técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código, representado na Figura 1.4 com uma linha pontilhada indicando um possível poder preditivo das abordagens em relação aos resultados das tarefas de compreensão de código.

A partir dessas motivações, a Figura 2 apresenta a esquemática dos problemas abordados neste trabalho. As setas indicam o fluxo em que as problemáticas foram investigadas e resolvidas. A linha pontilhada indica uma investigação onde verifica se há uma relação preditiva dos indicadores com tarefas de compreensão. Os números representam cada problemática que foi investigada neste trabalho.

Figura 2: Esquema da problemática investigada.



Fonte: Elaborado pelo autor.

(1) Ausência de uma visão geral da literatura sobre a medida da carga cognitiva dos desenvolvedores na área de engenharia de software. Faltam estudos que categorizem sistematicamente as tecnologias e indicadores psicofisiológicos relacionadas à carga cognitiva em

engenharia de software. Além disso, uma visão geral sobre a carga cognitiva em engenharia de software também poderia incluir as técnicas de *machine learning* utilizadas e os conjuntos de dados utilizados nessa área, os propósitos para os quais a carga cognitiva foi investigada na engenharia de software, as tarefas e artefatos adotados pelos estudos para investigar a carga cognitiva, bem como o perfil dos participantes recrutados para participar de experimentos envolvendo a carga cognitiva na engenharia de software.

Porém, as revisões da literatura relacionadas à carga cognitiva na área engenharia de software ignoraram esses aspectos (GUI et al., 2019; FAIRHURST; LI; COSTA-ABREU, 2017; BABLANI et al., 2019). Especificamente, GUI et al. (2019) sumarizaram os indicadores relacionadas às ondas cerebrais para sistemas de identificação de usuários. FAIRHURST; LI; COSTA-ABREU (2017) revisaram os indicadores psicofisiológicos que os pesquisadores aplicaram em técnicas de *machine learning* para classificar características pessoais, tais como, idade e gênero. BABLANI et al. (2019) apresentaram uma revisão da literatura sobre sensores e indicadores psicofisiológicos que poderiam ser aplicados às Interfaces Cérebro-Computador (BCI). Além disso, estudos negligenciaram a análise de aspectos relacionados à carga cognitiva na área de engenharia de software (KOSTI et al., 2018; FRITZ; MÜLLER, 2016; FAKHOURY et al., 2018). Portanto, é necessário realizar um levantamento da literatura de engenharia de software que contenha os principais aspectos relacionados à carga cognitiva. Desse modo, é importante classificar aspectos tais como, os conceitos utilizados para definir a carga cognitiva na engenharia de software, sensores, métricas, técnicas de *machine learning*, os propósitos da utilização da carga cognitiva na engenharia de software, artefatos e tarefas na qual a carga cognitiva foi abordada, perfis de participantes que foram recrutados para participar dos estudos, as metodologias de pesquisa que foram utilizadas para pesquisar carga cognitiva na engenharia de software, e os locais de publicação desses estudos. Esses aspectos são importantes porque fornecem aos pesquisadores uma visão geral do que foi considerado para medir a carga cognitiva na área de engenharia de software.

(2) Abordagens de EEG utilizadas na engenharia de software em tarefas de compreensão de código ignoraram a interação entre as diferentes regiões do cérebro na medição da carga cognitiva de desenvolvedores. CRK; KLUTHE (2016); YEH et al. (2017); DURAISINGAM; PALANIAPPAN; ANDREWS (2017); KOSTI et al. (2018); SIEGMUND et al. (2014) investigaram a carga cognitiva na área de engenharia de software através utilizando abordagens baseadas em dados cerebrais. SIEGMUND et al. (2014) utilizaram sensores de imagem fMRI através do indicador de *Blood Volume Pulse* (BVP), e demonstraram as áreas cerebrais em atividade durante as tarefas de compreensão do código. Este é um indicador de alta precisão espacial, pois evidencia a atividade em todo o espaço cerebral. Porém, também se trata de um sensor com baixa precisão temporal (MULERT; LEMIEUX, 2009), possuindo um tempo de resposta muito alto para ambientes que necessitam de resultados em tempo real. Por isso, estudos acadêmicos têm investigado abordagens para mensurar a carga cognitiva por meio de dispositivos com maior precisão temporal, tais como o EEG.

Pesquisas utilizaram várias abordagens baseadas em EEG para medir a carga cognitiva dos desenvolvedores. A partir dos sensores de EEG, as abordagens de CRK; KLUTHE (2016); DURAISINGAM; PALANIAPPAN; ANDREWS (2017) utilizaram indicadores tais como *Asymmetry Ratio* (ASR), *Event-Related Desyncrhonization* (ERD) e frequências de onda Alfa (α), Teta (θ), Beta (β) e Delta (δ). O problema é que essas abordagens de EEG utilizadas em pesquisas na área de engenharia de software para medir a carga cognitiva de desenvolvedores ignoraram (1) a interação entre a atividade de diferentes canais do EEG, (2) a definição de propriedades que as tornem específicas para tarefas de compreensão de código; (3) viabilizar de serem medidas sem depender de um período de descanso antes de cada tarefa, (4) e viabilizar o cálculo sem repetições amostrais da mesma tarefa. Esses aspectos não foram explorados e por isso é necessário desenvolver uma abordagem específica para mensurar a carga cognitiva em tarefas de compreensão de código. Estas características são importantes para viabilizar uma abordagem que calcule a carga cognitiva considerando a interação das várias regiões cerebrais envolvidas durante as tarefas de compreensão de código, desde que já foi constatado na literatura, o envolvimento da atividade neural em regiões cerebrais diferentes durante tarefas de compreensão de código (KARAS et al., 2021; PEITEK et al., 2021; SIEGMUND et al., 2017).

(3) Adoção de indicadores para medir carga cognitiva sem conhecimento empírico em relação a correlação dessas abordagens com as tarefas de compreensão de código. Pesquisas na área de engenharia de software (CRK; KLUTHE, 2016; LEE et al., 2017; DURAISINGAM; PALANIAPPAN; ANDREWS, 2017) ignoraram a análise de correlação entre carga cognitiva e aspectos das tarefas de compreensão de código. Dessa forma, abordagens foram utilizados sem nenhuma análise prévia de correlação com o propósito que foi investigado. Por exemplo, CRK; KLUTHE (2016) adotaram o *Event-Related Desyncronization* (ERD) para investigar a diferença do nível de conhecimento de desenvolvedores de software, mas não aplicaram testes de correlação com as tarefas de compreensão de código. KOSTI et al. (2018) utilizaram o *Phaze-Locking Value* (PLV) a partir das frequências de banda alfa (α), beta (β), delta (δ), theta (θ) e gamma (γ) do EEG para investigar o nível de dificuldade durante tarefas de programação, e não aplicaram testes de correlação com as tarefas de compreensão de código.

Deste modo, existem poucas análises sobre a correlação entre a carga cognitiva dos desenvolvedores com tarefas de compreensão de código. Além disso, esses trabalhos aplicaram indicadores sem qualquer adaptação ao contexto da compreensão de código, podendo talvez, perder potencial de correlação das abordagens utilizadas com tarefas de engenharia de software. Deste modo, faz-se necessário analisar a correlação desses indicadores com tarefas de engenharia de software. A falta dessa análise pode implicar em que pesquisadores continuem confiando em julgamentos próprios para escolher uma abordagem para mensurar a carga cognitiva em tarefas de engenharia de software.

(4) A utilização de dados EEG para classificar compreensão de código foi ignorada na literatura. Faltam estudos que utilizem dados de EEG em técnicas de *machine learning* para classificar compreensão de código dos desenvolvedores de software. O sensor de EEG possui

potencial de ser padrão em ambientes de engenharia de software devido à sua portabilidade, por possuir alta precisão temporal, e por coletar dados relacionados diretamente à atividade cerebral. Estudos utilizaram conjunto de dados com variados tipos de indicadores da carga cognitiva para treinar classificadores a propósitos que eram apenas relacionados à compreensão de software. Portanto, não trataram diretamente de classificar a compreensão de código dos desenvolvedores. Por exemplo, MÜLLER; FRITZ (2016) utilizaram técnicas de *machine learning* para classificar a presença de problemas de qualidade no código-fonte baseados em sensores de respiração, e de temperatura da pele. A investigação de LEE et al. (2017), a qual é o mais próximo deste problema, utilizou dados de EEG para classificar o nível de dificuldade nas tarefas de programação, e para classificar conhecimento dos desenvolvedores.

Além disso, outras pesquisas focaram em classificar tipos de emoções relacionadas em momentos de dificuldade em tarefas de engenharia de software (MÜLLER; FRITZ, 2015). Portanto, pouco se sabe quão efetivos são as técnicas de *machine learning* para classificar compreensão de código, utilizando dados a partir do EEG. Outro problema é que os estudos que treinaram técnicas de *machine learning* com dados EEG raramente verificaram se a efetividade das abordagens de *machine learning* é maior em relação a um classificador aleatório. A falta dessa verificação causa dúvidas sobre a real efetividade das técnicas de *machine learning* ao serem treinadas com dados EEG. Por fim, técnicas de *machine learning* também não foram treinadas com abordagens para calcular a carga cognitiva em tarefas de compreensão de código. Portanto, o impacto na efetividade desses classificadores ao serem treinados com uma abordagem específica a compreensão de código para calcular a carga cognitiva dos desenvolvedores é também desconhecida.

1.2 Objetivos e Questões de Pesquisa

Nas Seções anteriores foram apresentados os problemas identificados em relação ao uso da carga cognitiva na área de engenharia de software. Esta seção apresenta os objetivos (OB) e questões de pesquisa (QP) derivados dos problemas que foram identificados na Seção anterior.

OBGeral: o objetivo geral deste trabalho é investigar a relação das abordagens baseadas em EEG para mesurar a carga cognitiva de desenvolvedores de software em tarefas de compreensão de código. Em particular, foi escolhido abordagens de EEG para investigar a carga cognitiva devido à três principais motivos: (1) possui alta precisão temporal, ou seja, a resposta dos estímulos é instantaneamente refletida nos dados EEG, e (2) obtém dados relacionados a dinâmica neural. Com o objetivo de construir e avaliar essa abordagem, no início desta pesquisa foi realizada uma classificação geral, dos dispositivos, e indicadores relacionados à carga cognitiva nas tarefas na engenharia de software. Em um segundo momento, nas demais questões de pesquisa, a investigação focou nas tarefas de compreensão de código devido à dois principais motivos: (1) Trata-se de uma tarefa essencial na engenharia de software, pois, em várias tarefas de desenvolvimento de software tais como a revisão, e a manutenção de código, requer obrigatoriamente

que desenvolvedor compreenda o código; (2) Foi pouco investigada nos estudos sobre carga cognitiva na engenharia de software. Com base no objetivo geral (OBGeral), a seguinte questão de pesquisa geral (QPGeral) foi definida:

> ***QPGeral:*** qual é a relação entre abordagens baseadas em EEG utilizadas para mensurar a carga cognitiva com tarefas de compreensão de software?

Os objetivos específicos (OE) definidos abaixo são atribuídos respectivamente a resolver os problemas de pesquisas definidos na Seção 1.1. Os objetivos (OE) buscam resolver os problemas apresentados, e convergir na resposta da questão de pesquisa geral.

OE1: **realizar uma classificação geral dos estudos sobre a aplicação da carga cognitiva na área de engenharia de software.** Este objetivo busca resolver o problema de pesquisa 1, que é a ausência de uma visão geral da literatura sobre a medida da carga cognitiva dos desenvolvedores na área de engenharia de software. Para atingir o objetivo específico OB1, e resolver este problema de pesquisa busca-se: (1) Selecionar os estudos contemporâneos em relação a medida da carga cognitiva no desenvolvimento de software. Esse objetivo visa fornecer uma lista representativa do atual estado da arte das pesquisas produzidas sobre esse tema. Esta lista busca ser representativa por ser composta pelos principais autores de estudos sobre a carga cognitiva na engenharia de software; (2) Classificar estudos sobre as medidas da carga cognitiva dos desenvolvedores de software. A lista de estudos selecionada foi classificada em relação a importantes aspectos da carga cognitiva na engenharia de software, tais como os conceitos de carga cognitiva utilizadas na engenharia de software, sensores utilizados, métricas, técnicas de *machine learning*, propósitos para utilizar a carga cognitiva na engenharia de software, atividades, tarefas, participantes e locais de publicação; (3) Fornecer uma visão geral sobre a utilização da carga cognitiva na engenharia de software. Gerar um esquema de classificação a partir das subcategorias de cada aspecto investigado; (4) Identificar futuros desafios de pesquisa. Delinear futuros desafios de pesquisa a partir da classificação e da visão geral. Para contemplar o objetivo específico 1 (OE1), o Capítulo 3 busca responder à questão de pesquisa 1 (QP1):

> ***QP1:*** *qual é o estado-da-arte das medidas de carga cognitiva na área de engenharia de software?*

OE2: **desenvolver uma abordagem para mensurar a carga cognitiva considerando a interação entre diferentes regiões cerebrais durante tarefas de compreensão de código baseada nos dados EEG.** O objetivo específico 2 (OE2) busca desenvolver a abordagem para mensurar carga cognitiva durante tarefas de compreensão de código. Este objetivo busca resolver o problema de pesquisa 2, i.e., a falta de uma abordagem para representar a carga cognitiva dos desenvolvedores em tarefas de compreensão, definido na Seção 1.1. Como mencionado anteriormente, abordagens de EEG para mensurar a carga cognitiva em estudos na área de engenharia de software foram geralmente adaptadas de outras áreas. Desse modo, o objetivo é desenvolver uma abordagem para mensurar a carga cognitiva na engenharia de software. Essa

abordagem busca implementar lacunas existentes nas abordagens atuais tais como: a falta de considerar a interação entre diferentes canais de EEG, e a falta de propriedades para tornar a técnica específica para tarefas de compreensão de código. Em particular, a abordagem desenvolvida neste trabalho chama-se CogEff. Esta abordagem tem o objetivo de: (1) Conter um processo para mesurar a carga cognitiva dos desenvolvedores na engenharia de software em tarefas de compreensão de código; (2) Estabelecer um processo para filtrar sinais dos canais EEG; (3) Definir propriedades para estabelecer relevância que certas conectividades entre regiões possuem em uma tarefa especifica na engenharia de software; (4) Retornar um valor considerando a conectividade entre sensores EEG para ser associado a carga cognitiva. Para cumprir o objetivo específico 2 (OE2), o Capítulo 4 busca responder à questão de pesquisa 2 (QP2):

QP2: *como propor uma abordagem baseada em dados EEG para mensurar esforço cognitivo em tarefas de compreensão de código?*

OE3: **analisar a correlação das abordagens de carga cognitiva baseadas em dados EEG com aspectos das tarefas de compreensão de código.** O objetivo específico 3 (OE3) busca correlacionar abordagens atuais baseadas em EEG, e o abordagem desenvolvida neste trabalho, CogEff, com aspectos de compreensão de código. Este objetivo busca resolver o problema de pesquisa 3, i.e., a falta de análises sobre a relação das abordagens de carga cognitiva com tarefas da engenharia de software. Os atuais trabalhos sobre carga cognitiva na engenharia de software deliberadamente adaptaram abordagens para analisar carga cognitiva baseados em dados EEG no contexto da compreensão de código. Praticamente, as investigações de CRK; KLUTHE (2016); LEE et al. (2017); DURAISINGAM; PALANIAPPAN; ANDREWS (2017) foram conduzidas sem analisar a real correlação dos indicadores baseados em EEG com a compreensão de código. Especificamente, o objetivo específico 3 busca realizar: (1) a correlação das abordagens de EEG com compreensão de código. Essa avaliação inclui a abordagem CogEff, e as abordagens de EEG tradicionais alfa (α), beta (β), delta (δ), gamma (γ), theta (θ), *Asymmetry Ratio* (ASR), e *Event-Related Desyncronization* (ERD) com as respostas dos desenvolvedores; (2) a correlação dessas abordagens com o tempo decorrido com a tarefa de compreensão; (3) analisar a diferença dos valores da CogEff entre tarefas de compreensão de código com níveis de complexidades diferentes. Para alcançar o objetivo específico 3 (OE3), o Capítulo 5 busca investigar a correlação dos indicadores baseados em EEG com tarefas de compreensão de código, respondendo à questão de pesquisa 3 (QP3):

QP3: *qual a correlação das abordagens de carga cognitiva baseadas em dados EEG com a compreensão de código?*

OE4: **analisar a efetividade das técnicas de *machine learning* para classificar compreensão de código treinadas com dados de EEG.** O objetivo específico 4 (OE4) visa analisar o quão efetivos são as técnicas de *machine learning* para classificar compreensão de código caso forem treinadas com dados de EEG. Especificamente, o objetivo específico 4 refere-se a

resolver o problema de pesquisa 4, apontado na Seção 1.1, o qual aponta a ausência da investigação sobre a utilização de abordagens de EEG relacionados à carga cognitiva para classificar compreensão de código. Dessa forma, este trabalho busca: (1) investigar a efetividade das técnicas de *machine learning* para classificar compreensão de código dos desenvolvedores usando indicadores psicofisiológicos de EEG. Foram treinadas técnicas, com dados EEG, de *machine learning* geralmente utilizadas em aplicações *Brain Computer Interface* e adotadas nas pesquisas envolvendo *machine learning* com carga cognitiva na engenharia de software (FUCCI et al., 2019; LEE et al., 2017; MÜLLER; FRITZ, 2016). Especificamente, as técnicas treinadas com dados de EEG de desenvolvedores são *K-Nearest Neighbor* (KNN), *Neural Network* (NN), *Naïve Bayes* (NB), *Random Forest* (RF), e *Support Vector Machine* (SVM); (2) investigar se a efetividade dessas técnicas de *machine learning* são significativamente mais altas que um classificador aleatório; (3) investigar se adicionar os dados gerados pela abordagem desenvolvida nesse trabalho, CogEff, ao conjunto de dados de EEG causa melhora na efetividade da classificação de compreensão de código dos desenvolvedores; (4) discutir as diferentes características das técnicas de *machine learning* em relação ao desempenho aos dados EEG. Para contemplar o objetivo específico 4 (OE4), o Capítulo 6 busca analisar a efetividade das técnicas de *machine learning* treinados com EEG para classificar a compreensão de código dos desenvolvedores, respondendo à questão de pesquisa 4 (QP4), a qual é formulada a seguir:

QP4: *quão efetivas são as técnicas de machine learning para classificar compreensão de código usando dados de EEG?*

1.3 Metodologia

Esta Seção apresenta as metodologias utilizadas para responder as questões de pesquisa definidas neste trabalho. As metodologias utilizadas são apresentas a seguir.

Estudo de mapeamento sistemático. A primeira questão de pesquisa (QP1) aponta sobre a falta de uma classificação do estado da arte sobre carga cognitiva na engenharia de software. Para isso, este trabalho adotou o método de Estudo de Mapeamento Sistemático. Esta metodologia é conhecida na literatura por ter diretrizes empíricas bem definidas (KITCHENHAM; CHARTERS, 2007; KITCHENHAM; BUDGEN; PEARL BRERETON, 2011; GONCALES et al., 2019). Essas diretrizes consistem na definição de uma string de busca, o qual é composta por termos presentes nos estudos que são referência da área de pesquisa. Definição de bases de busca relacionada a área de pesquisa, isto é, relacionadas a ciência da computação e engenharia de software. A string de busca definida sistematicamente é utilizada nessas bases de busca. Em seguida, definição de critérios para inclusão e exclusão de estudos. Aplicação desses critérios em um processo de seleção de oito etapas, incluindo *snowballing*. O estudo de mapeamento sistemático é apresentado no Capítulo 3.

Proposta de solução. A segunda questão de pesquisa (QP2) desenvolve uma abordagem,

chamada de CogEff, para mensurar a carga cognitiva na engenharia de software, especificamente, durante tarefas de compreensão de código. Esta abordagem utiliza dados EEG para mensurar a carga cognitiva. A partir disso, a abordagem CogEff estabelece (1) um processo para medir a carga cognitiva na engenharia de software em tarefas de compreensão de código; (2) define um processo de filtragem de dados EEG. Trata-se de um filtro FIR que realiza janelamentos *Hamming* nos dados EEG, com *High* e *Low-pass filter*. É um filtro bem estabelecido na literatura com limites de corte para remover sinais que não interessam ao contexto de compreensão de código; (3) estabelece conexões entre dois canais EEG. Essas conexões são estabelecidas através de uma abordagem *over-time*; (4) define propriedades para definir a relevância de específicas conexões em relação à tarefa que está sendo investigada. Os pesos podem ser atribuídos de forma *adhoc*. Essa característica torna a técnica suscetível à adaptação à novas evidências na literatura; (5) retorna um valor final o qual é definido pela densidade do grafo que foi formado através das conexões entre os canais de EEG.

Estudo empírico. A terceira questão de pesquisa (QP3) avalia a correlação das abordagens utilizadas para mensurar a carga cognitiva utilizando indicadores de EEG em relação à compreensão de código. Nesta avaliação, também é inclusa a abordagem desenvolvida nesse trabalho, a CogEff. Portanto, a avaliação da abordagem desenvolvida neste trabalho segue a mesma metodologia. Especificamente, foram aplicados testes de correlação entre as abordagens e aspectos específicos da compreensão de software, tais como, as respostas das questões de compreensão de programas, e o tempo para concluir as tarefas de compreensão. Deste modo, para analisar a correlação entre as abordagens com a compreensão de código foram realizados testes de: (1) correlação de *Kendall*, e *Spearman* das abordagens com as respostas das tarefas de compreensão; (2) correlação de *Pearson* e *Spearman* entre as abordagens com o tempo para concluir as tarefas de compreensão. Por fim, é realizado um teste com a abordagem CogEff. É analisada a diferença entre os resultados da CogEff entre tarefas de complexidade baixa e alta. Para isso, as tarefas de complexidade baixa e alta foram categorizadas em relação à complexidade ciclomática dessas tarefas. Para testar se a diferença é significativa foram aplicados o *t-test*, e Wilcoxon.

Estudo empírico. A quarta questão de pesquisa (QP4) investiga a utilização de abordagens de EEG para classificar compreensão de código. Além disso, também investiga se as técnicas de *machine learning* são significativamente mais efetivas que um classificador aleatório, e finalmente, se a adição dos dados da CogEff ao conjunto de dados de treinamento afeta a efetividade dos classificadores. Dados das ondas cerebrais tais como, as potências de bandas alfa (α), theta (θ), beta (β), delta (δ) e gama (γ) foram utilizados para treinar técnicas de *machine learning*. Essas técnicas foram treinadas para classificar a compreensão de código. A avaliação especificamente consiste em uma discussão dos resultados em relação às métricas de efetividade tais como a precisão, *recall*, e *f-measure*. As técnicas de *machine learning* implementadas são a *K-Nearest Neighbor* (KNN), *Neural Networks* (NN), *Support Vector Machine* (SVM), *Naïve Bayes* (NB), *Random Forest* (RF), e um classificador aleatório. Especificamente, para esta ava-

liação são realizadas (1) uma descrição dos resultados da precisão, *recall*, e *f-measure*. Além disso, foi avaliado se a efetividade das técnicas de *machine learning*, em termos da *f-measure* é maior do que 80%. Para isso, foi utilizado o *t-test* de uma amostragem; (2) o mesmo teste foi utilizado para cada classificador, para testar se as amostras de efetividade de *f-measure* tem média superior a um classificador aleatório; (3) foram aplicados o *t-test* e Wilcoxon para avaliar se existe diferença entre a efetividade dos classificadores que foram treinados apenas com os abordagens tradicionais de EEG de carga cognitiva em relação aos classificadores que foram treinados com o conjunto de dados contendo a abordagem CogEff.

Por fim, no geral, as metodologias deste trabalho consistem em viabilizar: (1) o estado da arte em relação a utilização das medidas da carga cognitiva na engenharia de software; (2) uma abordagem para representar esforço mental durante tarefas de compreensão de código; (3) avaliação da correlação da CogEff e de abordagens de EEG tradicionais em relação a aspectos relacionados a compreensão de código; e uma (4) avaliação da efetividade de técnicas de *machine learning* treinados com dados de EEG para classificar a compreensão de código. Na próxima seção, as contribuições obtidas a partir da execução dessa etapa metodológica são descritas.

1.4 Contribuições

Esta seção descreve as contribuições (CT) esperadas desta pesquisa. São esperadas as seguintes contribuições:

(1) Estado da arte sobre as medidas da carga cognitiva em engenharia de software: Através de um estudo de mapeamento sistemático é esperado obter uma visão geral sobre a carga cognitiva na engenharia de software. A partir deste estudo, espera-se que seja realizada (1) uma categorização do estado da arte em relação aos conceitos de carga cognitiva mencionados nos estudos de engenharia de software, sensores, métricas, técnicas de *machine learning*, propósitos da utilização da carga cognitiva na engenharia de software, tipos de artefatos e tarefas que foram utilizados para medir a carga cognitiva, a quantidade de participantes recrutados, os métodos de pesquisa utilizados para investigar a carga cognitiva na engenharia de software, e os locais de publicação; (2) obtenção de uma visão geral do estado da arte sobre as medidas da carga cognitiva na engenharia de software através de um esquema de classificação; (3) obter uma lista de estudos que exploraram a carga cognitiva na engenharia de software através de sensores biométricos; (4) listar as principais contribuições e desafios de pesquisa que podem ser explorados pela academia e indústria.

(2) Uma abordagem para mensurar carga cognitiva em tarefas de compreensão de código: espera-se o desenvolvimento de uma abordagem para mensurar a carga cognitiva dos desenvolvedores durante tarefas de compreensão de código. Especificamente, é esperado que essa abordagem contribua com: (1) um filtro adaptado para ser aplicado a dados EEG coletados durante tarefas de compreensão de código; (2) a medida da carga cognitiva através do

estabelecimento de conexões entre diferentes canais EEG utilizando uma abordagem *over-time*; (3) propriedades que podem ser aplicadas as conexões; (4) algoritmos para calcular a carga cognitiva.

(3) Uma avaliação da correlação entre a abordagem CogEff e abordagens tradicionais de EEG em tarefas de compreensão de código: é esperado uma análise da correlação da abordagem desenvolvida nesse trabalho e abordagens tradicionais baseados no EEG em relação a tarefas de compreensão de código. Em particular, espera-se: (1) uma demonstração de como correlacionar abordagens para analisar carga cognitiva com tarefas na engenharia de software. Por exemplo, correlação de abordagens tradicionais de EEG com aspectos psicométricos da carga cognitiva tais como o tempo decorrido para concluir a tarefa de compreensão, e as respostas das tarefas de compreensão; (2) a evidência de que existe correlação da abordagem desenvolvida neste trabalho, e das abordagens tradicionais baseadas em EEG com tarefas de compreensão de código; (3) evidência da diferença no esforço cognitiva entre tarefas de complexidade alta e complexidade baixa.

(4) Avaliação da efetividade das técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código: espera-se uma avaliação da efetividade das técnicas de *machine learning* treinadas com um conjunto de dados de dados EEG para classificar compreensão de código. Em particular espera-se as seguintes contribuições: (1) classificares de compreensão de código através dos dados de EEG; (2) uma metodologia para realizar o pré-processamento de dados EEG para treinar técnicas de *machine learning* com dados EEG; (3) uma metodologia empírica para analisar a efetividade de técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código; (4) comparação da efetividade das técnicas de *machine learning* para classificar a compreensão de código baseado em dados de EEG; (5) discussão de desafios e trabalhos futuros em relação a utilização de dados EEG em técnicas de *machine learning* para classificar compreensão de código.

1.5 Estrutura

Esta tese está estruturada nos seguintes capítulos: Capítulo 2 descreve conceitos básicos do trabalho. Capítulo 3 apresenta os trabalhos relacionados que foram selecionados através de um protocolo de mapeamento sistemático. Capítulo 4 apresenta a abordagem CogEff. Capítulo 5 apresenta a avaliação da correlação entre indicadores psicofisiológicos baseados em EEG e aspectos das tarefas de compreensão de código. Capítulo 6 apresenta a avaliação da efetividade das técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código. Capítulo 7 descreve as considerações finais dessa tese.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta conceitos e teorias relacionados a esta pesquisa. A Seção 2.1 apresenta a definição de carga cognitiva. A Seção 2.1 descreve implicações do nível da carga cognitiva nos desenvolvedores. A Seção 2.3 apresenta o processo que é geralmente utilizado pela literatura para medir a carga cognitiva na engenharia de software. A Seção 2.4 apresenta os conceitos básicos dos sinais do eletroencefalograma. Por fim, a Seção 2.5 apresenta as considerações finais deste capítulo.

2.1 Carga cognitiva

Carga cognitiva refere-se ao nível de esforço mental que os usuários utilizam para executar tarefas cognitivas (MÜLLER; FRITZ, 2015). Por exemplo, na engenharia de software, desenvolvedores de software gastam esforço cognitivo para compreender artefatos de software, tais como código fonte, modelos UML, diagramas arquiteturais, entre outros. Estudos recentes apontam que uma carga cognitiva alta está associada à uma alta carga na memória de trabalho dos desenvolvedores, evidenciando que desenvolvedores executaram tarefas não triviais. Por outro lado, carga cognitiva baixa está associada à baixa carga de informações na memória de trabalho dos desenvolvedores, implicando que está lidando com tarefas de fácil entendimento.

- *Carga cognitiva intrínseca:* refere-se ao esforço mental que varia exclusivamente a característica inerente da tarefa cognitiva (SWELLER, 2010). Desta forma, é a carga cognitiva que os usuários utilizam referente ao nível de complexidade da tarefa;

- *Carga cognitiva extra:* está relacionada ao esforço mental a instruções ou artefatos redundantes, que praticamente foram acrescentados e informam a mesma informação passada anteriormente (SWELLER, 2010). Informações redundantes implicam em esforço extra, sem acrescentar informação nova ao que está sendo interpretado;

- *Carga cognitiva pertinente:* refere-se à concepção de construção dos esquemas básicos de conhecimento (SWELLER, 2010). Isto é, o conhecimento básico dos usuários em relação a um domínio. Uma vez que um usuário utiliza a carga cognitiva para adquirir esquemas básicos, este recurso não é usado novamente. Por exemplo, usuários não gastam carga cognitiva pertinente novamente para descobrir o papeis de estruturas já assimiladas.

2.2 Implicações da carga cognitiva nos desenvolvedores

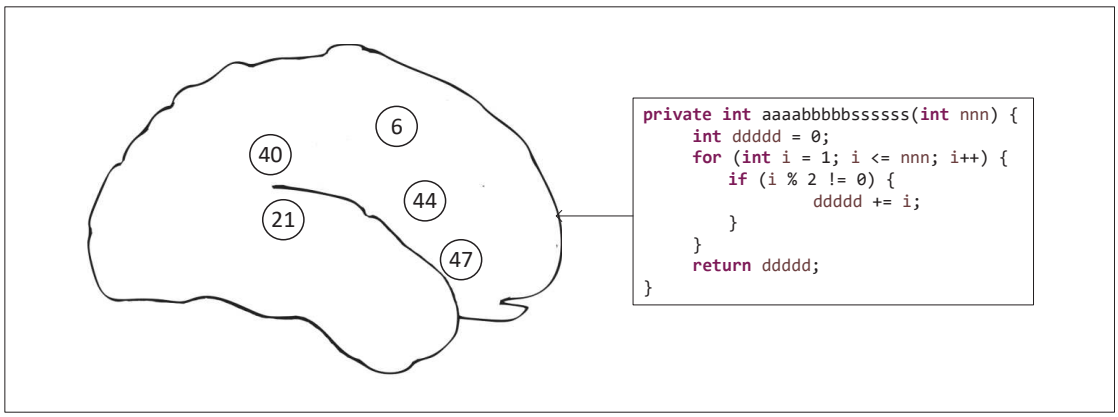
Na engenharia de software, é possível analisar os sinais dos níveis de carga cognitiva através de várias manifestações fisiológicas de desenvolvedores. Estudos anteriores já evidenciaram que os indicadores do nível de carga cognitiva estão relacionados a manifestações, tais como

as da atividade cerebral (CRK; KLUTHE, 2016; SIEGMUND et al., 2017), temperatura da pele e frequência cardíaca (ZÜGER; FRITZ, 2018). Através da análise de atividades de ondas cerebrais, CRK; KLUTHE (2016) apontaram que o aumento da carga cognitiva se correlaciona com menores níveis de expertise dos desenvolvedores de software. SIEGMUND et al. (2017) chegou à mesma conclusão, mas analisando as atividades dos fluxos de oxigenação do sangue no cérebro. Os autores concluem que algumas regiões do cérebro são desativadas à medida que seu nível de concentração aumenta. Além disso, SIEGMUND et al. (2017) atribuíram as regiões do cérebro que são ativadas enquanto os desenvolvedores analisam tarefas de compreensão de código.

As ativações foram atribuídas às áreas de Broadmann (BA) específicas. As áreas de Broadmann definem 52 regiões específicas do córtex cerebral com identificadores. A Figura 3 apresenta algumas áreas de Broadmann delineadas enquanto os desenvolvedores compreendem o código. As áreas do cérebro responsáveis pela memorização das palavras e valores numéricos (BA 6 e 40) e responsáveis pelo processamento da linguagem (BA 21 e 44) foram ativadas. Consequentemente, ocorre um *chunk*, isto é, uma integração de informações que atribui semântica às operações (BA 21, 44 e 47).

A Figura 3 apresenta o cérebro do desenvolvedor processando um artefato de software sem identificadores legíveis. A tarefa é trivial, mas os níveis de carga cognitiva aumentaram, tornando a tarefa não mais trivial. Assim, considerando a perspectiva da carga cognitiva, a percepção real do desenvolvedor é que a tarefa tem um alto nível de dificuldade. Neste exemplo, áreas específicas foram ativas durante as tarefas de compreensão de código. Assim, a carga cognitiva dos desenvolvedores tem potencial de ser explorada em relação às tarefas de engenharia de software. Para isso, é importante investigar quais aspectos são relevantes para medir a carga cognitiva na engenharia de software, tais como sensores utilizados, métricas, técnicas de classificação, propósitos, artefatos e tarefas de software.

Figura 3: Áreas de Broadmann do cérebro que são ativadas durante a compreensão do código.

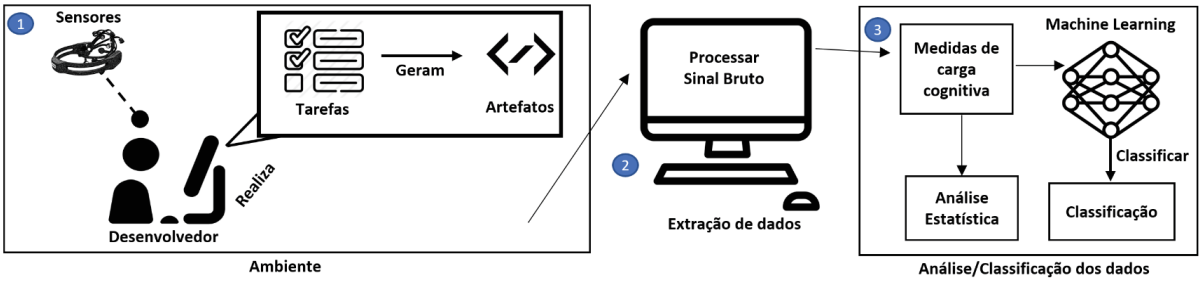


Fonte: Elaborado pelo autor.

2.3 Processo de mensuramento da carga cognitiva na Engenharia de Software

A Figura 4 apresenta uma visão geral dos processos utilizados em trabalhos anteriores (LEE et al., 2017; FUCCI et al., 2019; MÜLLER; FRITZ, 2016) para mensurar carga cognitiva. Especificamente, o processo para mensurar carga cognitiva é composto por três partes, as quais são descritas a seguir:

Figura 4: Processo de medida da carga cognitiva na engenharia de software.



Fonte: Elaborado pelo autor.

Em geral, mensurar a carga cognitiva consiste em um processo de três etapas principais:

Etapa 1. Ambiente de desenvolvimento: consiste no ambiente onde desenvolvedores de software desempenham as tarefas de engenharia de software com dispositivos coletando dados psicofisiológicos. Nesta parte do processo, desenvolvedores realizam suas tarefas, e partir delas geram artefatos de software. Enquanto isso, os sinais psicofisiológicos são monitorados através de vários tipos de dispositivos. Desse modo, sinais psicofisiológicos tais como os sinais EEG, ou a taxa de fixação dos olhos são capturadas. Estes sinais, por exemplo, são capturados através de um eletroencefalograma e de um *Eye-tracker* respectivamente. Esses sinais são considerados brutos porque existem neles informações que também não são relacionadas diretamente ao evento da atividade de software realizada.

Etapa 2. Extração de dados: compreende em um servidor, ou computador, que processa os dados psicofisiológicos coletados enquanto os desenvolvedores conduzem as tarefas experimentais. Essa parte do processo é responsável por processar o sinal bruto dos sensores com a finalidade de realizar um tratamento dos dados, e assim filtrar esses ruídos. Desse modo, informações que não são relacionadas ao esforço mental devem ser descartadas, i.e., que não tem relação com a atividade de desenvolvimento. Por exemplo, sinais que representam movimentos involuntários. Essas informações são conhecidas como ruídos no sinal. Em seguida, após a remoção dos ruídos, pesquisadores costumam extrair métricas a partir desses sinais, tais como, a média da potência das ondas cerebrais, ou a média do piscar dos olhos.

Etapa 3. Análise/Classificação dos dados: consiste em um programa de computador ou procedimentos de análise, que moldam o propósito e o motivo para utilização dos dados psicofisiológicos coletados a partir dos desenvolvedores de software. Esta parte do processo é responsável por analisar e classificar os dados. As métricas geradas no processamento são cor-

relacionadas a algum propósito relacionado a engenharia de software. Para isso, um conjunto de técnicas de aprendizado de máquina tem finalidade de correlacionar e classificar as métricas a um propósito, tais como, o nível de produtividade dos desenvolvedores, o nível de dificuldade para desempenhar uma tarefa, ou classificar a compreensão do código fonte.

2.4 Eletroencefalografia

A eletroencefalografia é uma técnica não invasiva que possibilita mensurar a dinâmica eletrofisiológica produzidas pela interação dos neurônios (COHEN, 2017). A dinâmica desses sinais constitui nas ondas cerebrais. Essas ondas cerebrais podem ser capturadas por um eletroencefalograma (EEG) (MAIORANA; SOLÉ-CASALS; CAMPISI, 2016). O ato de monitorar as ondas cerebrais com algum tipo de eletroencefalograma é chamado eletroencefalografia. A eletroencefalografia pode ser intrusiva ou não. Neste trabalho, foram capturados dados usando um método não invasivo, em particular usando um dispositivo EEG sem fio portátil chamado Emotiv Epoc+ (Emotiv Systems, 2021). Este é um dispositivo que é composto por quatorze canais. Os eletrodos deste dispositivo estão organizados em relação ao sistema internacional 10-20. Este é o método mais simples de coletar ondas cerebrais em relação aos métodos invasivos, os quais requerem intervenção cirúrgica para a implantação eletrodos. A principal vantagem do método intrusivo é a maior resolução nos dados, além de não conter ruídos que são captados no escalpo.

Os eletrodos em contato com o escalpo são sensíveis a ruídos, tais como, movimentos dos músculos, sinais de interferência de outros equipamentos eletrônicos e movimentos locomotores. Todos esses ruídos são contidos e capturados pelo EEG e reunidos nas ondas cerebrais. Desse modo, é importante realizar a filtragem dos dados das ondas cerebrais. Além disso, cinco grupos principais de frequências de ondas compõem as ondas cerebrais: delta (δ) [1-4Hz], theta (θ) [4-8Hz], alfa (α) [8-13Hz], beta (β) [13-30Hz] e gamma (γ) [30-70Hz] (MAIORANA; SOLÉ-CASALS; CAMPISI, 2016). Cada uma dessas bandas de frequência de ondas cerebrais possui uma faixa e amplitude de frequência específica e exibe mais ou menos atividade sob certas circunstâncias. As ondas são apresentadas a seguir (MAIORANA; SOLÉ-CASALS; CAMPISI, 2016):

- Delta (δ) [1-4Hz]: essa onda não apresenta oscilação quando desenvolvedores de software estão em atividade;
- Theta (θ) [4-8Hz]: esta onda está presente quando alguma atividade é realizada;
- Alfa (α) [8-13Hz]: a onda alfa está relacionada à ausência de atividade;
- Beta (β) [13-30Hz]: a atividade dessa onda indica estado e foco de alerta;
- Gama (γ) [30-70Hz]: a atividade dessa onda indica a concentração do usuário.

2.5 Considerações finais

Este capítulo apresentou os conceitos básicos relacionados a este trabalho. Especificamente, três conceitos básicos foram apresentados: conceitos básicos sobre a teoria de carga cognitiva (CLT) utilizada na área da psicologia, as implicações da carga cognitiva na área de engenharia de software, conceitos sobre o processo para medir a carga cognitiva na engenharia de software, e os sinais do eletroencefalograma. Trata-se de conceitos relacionados a uma visão geral sobre as medidas da carga cognitiva na engenharia de software.

Foi descrita a teoria tradicional de carga cognitiva, a qual é utilizada nos estudos de psicologia, como a carga cognitiva é mensurada nos estudos de engenharia de software, as implicações da carga cognitiva nos desenvolvedores, e também foi descrito no que consiste o sinal psicofisiológico utilizado neste trabalho, isto é, o eletroencefalograma. O próximo capítulo investiga sobre medidas da carga cognitiva na engenharia de software através de um estudo de mapeamento sistemático. Especificamente, o capítulo seguinte realiza uma classificação geral e sumarização dos principais estudos dessa área de pesquisa.

3 MAPEAMENTO SISTEMÁTICO DA LITERATURA

Este capítulo tem o objetivo de classificar os trabalhos relacionados a esta tese. Para isso, um mapeamento sistemático da literatura foi realizado, visando responder à questão de pesquisa 1 (QP1). Um protocolo de mapeamento sistemático da literatura foi elaborado baseado em diretrizes da literatura (KITCHENHAM; CHARTERS, 2007; KITCHENHAM; BUDGEN; PEARL BRERETON, 2011). Especificamente, este capítulo tem dois objetivos: (1) classificar e fornecer uma visão geral dos estudos produzidos em relação a medida da carga cognitiva dos desenvolvedores, e (2) identificar desafios e trabalhos futuros através da literatura selecionada. Por esse motivo, um Estudo de Mapeamento Sistemático (SMS) foi realizado com base em um protocolo de seleção (KITCHENHAM; BUDGEN; PEARL BRERETON, 2011; PETERSEN; VAKKALANKA; KUZNIARZ, 2015; KITCHENHAM; CHARTERS, 2007; GONCALES; FARIAS; SILVA, 2021; BISCHOFF et al., 2019; GONÇALES et al., 2019). Uma busca inicial em onze bases de dados encontrou um total de 4.175 estudos candidatos a serem selecionados como estudos primários. Um processo de filtragem desses estudos foi conduzido e um total de 63 artigos foram selecionados como estudos primários, os quais (1) atendiam às questões de pesquisa definidas, (2) seguiram os critérios de inclusão e exclusão. Os resultados são analisados e discutidos no restante deste capítulo.

O restante deste capítulo está organizado nas seguintes seções: A Seção 3.1 apresenta a metodologia de mapeamento sistemático adotada para selecionar os estudos mais avançados. A Seção 3.2 resume os resultados respectivos para as questões de pesquisa desta pesquisa. A Seção 3.3 apresenta uma discussão adicional sobre os resultados e propõe desafios para pesquisas futuras. A Seção 3.4 resume os trabalhos relacionados a este mapeamento sistemático, discutindo os pontos em comum. A Seção 3.5 descreve as medidas adotadas para evitar viés na validade na seleção dos estudos bem como nos resultados desta pesquisa. A Seção 3.6 apresenta a lista de trabalhos relacionados à abordagem CogEff. A Seção 3.7 realiza uma análise comparativa entre os trabalhos relacionados com a CogEff. Finalmente, a Seção 3.8 discute as considerações finais e os desafios futuros deste mapeamento sistemático.

3.1 Metodologia

Esta seção descreve a metodologia de mapeamento sistemático da literatura (SMS) que foi utilizada para realizar o levantamento de trabalhos relacionados. A Seção 3.1.1 define as questões de pesquisa deste estudo. A Seção 3.1.2 define a estratégia de pesquisa, como as *strings* de busca e os motores de busca. A seção 3.1.3 especifica os critérios usados para incluir e excluir estudos pelo processo de seleção. Finalmente, a Seção 3.1.4 descreve as etapas aplicadas no processo de filtragem de estudos.

3.1.1 Objetivos de pesquisa

Este estudo de mapeamento sistemático tem dois objetivos principais: (1) classificar os estudos primários em relação a utilização da carga cognitiva de desenvolvedores na engenharia de software; e (2) apontar novos desafios para pesquisas futuras. Para atingir esses objetivos, as seguintes questões de pesquisa (QP) exploraram e classificaram os principais fatores e métodos envolvidos para medir a carga cognitiva (QP1-QP8), e os aspectos estatísticos sobre a pesquisa em carga cognitiva na engenharia de software, como os métodos de pesquisa utilizados (QP9) e locais de publicação (QP10). Este estudo investiga esses aspectos porque representam uma visão geral da utilização da carga cognitiva na área de engenharia de software. Além disso, a seção de discussão destaca desafios sobre as medidas de carga cognitiva em engenharia de software. A Tabela 1 descreve as questões de pesquisa que foram investigados nesse mapeamento sistemático, juntamente com suas motivações e variáveis exploradas.

A Tabela 1 descreve as questões de pesquisa (QPs) deste mapeamento sistemático. Além disso, a Tabela 1 também descreve as motivações e as dimensões das questões de pesquisa.

Tabela 1: Questões de pesquisa definidas

Questões de Pesquisa (RQ's)	**Motivação**	**Dimensão**
QP1: como a carga cognitiva foi definida na literatura de engenharia de software?	A carga cognitiva é um termo mal utilizado na engenharia de software. Esta questão de pesquisa mapeia quais as definições foram utilizadas sobre carga cognitiva na engenharia de software.	Carga Cognitiva
QP2: quais são os tipos de sensores adotados pelos estudos para medir a carga cognitiva?	Classificar os tipos de sensores usados pelos estudos primários para coletar dados relacionados à carga cognitiva dos desenvolvedores.	Sensores
QP3: quais métricas foram usadas para medir a carga cognitiva dos desenvolvedores?	Identificar as métricas que foram correlacionadas com a carga cognitiva dos desenvolvedores.	Métricas
QP4: quais técnicas de *machine learning* foram usados para classificar a carga cognitiva de desenvolvedores?	Descobrir as principais técnicas de *machine learning* adotadas para classificar a carga cognitiva.	Algoritmos
QP5: para quais propósitos a carga cognitiva dos desenvolvedores foi medida?	Descobrir quais propósitos os estudos primários mediram a carga cognitiva.	Propósitos
QP6: quais tarefas foram usadas para medir a carga cognitiva dos desenvolvedores?	Identificar quais tarefas que a literatura atual de engenharia de software utilizou para medir a carga cognitiva dos desenvolvedores.	Tarefas de Desenvolvimento de Software
QP7: quais artefatos foram usados nas tarefas cognitivas?	Entender e investigar quais artefatos os estudos adotaram para executar tarefas experimentais para medir a carga cognitiva dos desenvolvedores.	Artefatos
QP8: quantos participantes os estudos recrutaram para medir a carga cognitiva dos desenvolvedores?	Classificar o número de participantes que geralmente eram recrutados pelos pesquisadores para realizar os experimentos sobre medidas de carga cognitiva na engenharia de software.	Número de participantes
QP9: quais métodos de pesquisa foram utilizados para investigar a carga cognitiva nas tarefas de desenvolvimento de software?	Classificar os métodos de pesquisa utilizadas pela literatura atual para investigar a carga cognitiva dos desenvolvedores.	Métodos de pesquisa
QP10: onde os estudos foram publicados?	Identificar os locais de publicação nas quais as pesquisas que mediram a carga cognitiva na engenharia de software foram publicadas.	Locais de publicação

Fonte: Elaborado pelo autor.

3.1.2 Estratégia de busca

A Seção 3.1.2.1 apresenta a combinação de palavras-chave que forma a *string* de Busca. A Seção 3.1.2.2 lista as bases de dados utilizadas para pesquisar pelos estudos.

3.1.2.1 Construção da String de Busca (SS)

Esta Seção apresenta os passos da construção da *string* de busca, i.e., os termos utilizados nos motores de busca para pesquisar pelos estudos. A *string* de busca é composta por um conjunto de termos que delimitam o escopo e propósito deste estudo sistemático. Nesta pesquisa, foram identificados os termos de busca utilizando o método PICo (População, Intervenção e Contexto), o qual é sugerido para ser utilizado em estudos qualitativos (DAVIES, 2011; HUANG; LIN; DEMNER-FUSHMAN, 2006). Este método foi utilizado em estudos de revisão sistemática para definir os termos de busca (KITCHENHAM; MENDES; TRAVASSOS, 2007; QIU et al., 2014). A literatura defende em dividir as questões de pesquisa em aspectos individuais relacionado ao assunto investigado (PETERSEN; VAKKALANKA; KUZNIARZ, 2015; KITCHENHAM; MENDES; TRAVASSOS, 2007). Estes aspectos implicam em uma lista a qual contêm diferentes palavras chaves e sinônimos. Combinando estes termos a partir dessa lista utilizando booleanos tais como “ANDs” e “ORs” resulta em uma *string* de busca. O método PICo consiste em estruturar a *string* de busca em aspectos de Populações, Intervenções, e Contexto. Estes aspectos são detalhados a seguir.

As Populações referem-se aos termos relacionados a padrões e tecnologias. Para este aspecto, foram definidos os termos *devices*, *sensors*, *BCI*, *EEG*, *ECG*, *EDA*, e *fMRI*. Estes são termos que foram encontrados na literatura que mede a carga cognitiva na engenharia de software (CRK; KLUTHE, 2016; MÜLLER; FRITZ, 2016; PEITEK et al., 2018a). A população não foi limitada a desenvolvedores de software, pois os estudos realizados em laboratórios podem recrutar pessoas que não são desenvolvedores para com o objetivo de usá-los como grupo de controle. As populações se concentraram apenas em sensores porque a engenharia de software construiu um novo paradigma para medir a carga cognitiva dos desenvolvedores com base nesses sensores.

As Intervenções são relacionadas aos termos que definem o propósito das tecnologias anteriormente mencionadas, i.e., as medidas de carga cognitiva na engenharia de software. Sinônimos, tais como *emotions*, *mental effort*, e *biometrics*, foram considerados. Não foram utilizados termos que são relacionados a carga cognitiva tais como estresse, falta de sono, e pressão no ambiente de trabalho para evitar em restringir a busca a estes domínios específicos. Foi considerado *emotion* como sinônimo da carga cognitiva devido à dois motivos. Primeiro, pesquisadores em engenharia de software adotaram métricas para medir a carga cognitiva para indicar emoções negativas e positivas (MÜLLER; FRITZ, 2016, 2015). Segundo, devido à evidência de que emoções afetam a carga cognitiva (PLASS; KALYUGA, 2019).

Os termos relacionados ao Contexto da *string* de busca contém os termos relacionados ao contexto das contribuições esperadas por praticantes e pesquisadores. Desta forma, este trabalho considera termos em relação ao contexto de engenharia de software tais como, *diagram*, *code*, *bugs*, *software development*, *software testing*, *software maintenance*, *computer programming*, *program comprehension*, *software deployment*, *software modeling*, e *software analysis*. Após as definições de termos principais e os sinônimos, os termos principais foram concatenados com operadores AND e os sinônimos foram concatenados com operadores OR.

De forma geral, foram realizadas as seguintes etapas para definir os termos de busca: (1) Definir termos primários utilizando PICo; (2) Identificar sinônimos, palavras relacionadas ou termos alternativos aos termos primários; (3) Verificar se principais artigos deste tópico de pesquisa contém as palavras chaves selecionadas; (4) Associar sinônimos, palavras alternativas, ou termos relacionados aos termos primários com o booleano "OR"; e, finalmente (5) Relacionar os termos primários com o *booleano* "AND". Tabela 2 apresenta os três termos principais e seus sinônimos.

Tabela 2: Definição das principais palavras chave e seus sinônimos que compõem a *string* de busca.

Termos principais	**Sinônimos**
Sensors	devices OR sensors OR BCI OR EEG OR ECG OR EDA OR fMRI
Cognitive Load	emotions OR "mental effort" OR biometrics
Software engineering	"software engineering" OR diagram OR code OR bugs OR "software development" OR "software testing" OR "software maintenance" OR "computer programming" OR "program comprehension" OR "software deployment" OR "software analysis"

Fonte: Elaborado pelo autor.

A seguinte *String* de busca (SS) é o resultado da combinação dos termos definidos e seus sinônimos. Esta *string* de busca foi aplicada nos motores de busca definidos na próxima seção.

(devices OR sensors OR BCI OR EEG OR ECG OR EDA OR fMRI) AND ("cognitive load" OR emotions OR "mental effort" OR biometrics) AND ("software engineering" OR diagram OR code OR "software development" OR "software testing" OR "software maintenance" OR "computer programming" OR "program comprehension" OR "software deployment" OR "software analysis")

3.1.2.2 Motores de Busca

Esta Seção define os motores de busca utilizados para pesquisar os estudos primários. Esses motores de busca foram escolhidos pois estudos anteriores (GONCALES et al., 2019; SHARAFI; SOH; GUÉHÉNEUC, 2015) mostraram a efetividade deles na realização de estudos de

mapeamento sistemático da literatura na área de engenharia de software. A Tabela 3 apresenta a lista dos motores de busca selecionados.

Tabela 3: Lista dos motores de busca selecionados

Search Engines	Link
ACM Digital Library	http://dl.acm.org/
CiteSeerX Library	http://citeseerx.ist.psu.edu/
Google Scholar	https://scholar.google.com.br/
IEEE Explore	http://ieeexplore.ieee.org/
Inspec	http://digital-library.theiet.org/
Microsoft Academic	https://academic.microsoft.com/
Pubmed	https://www.ncbi.nlm.nih.gov/pubmed/
Scopus	http://www.scopus.com/
Science Direct	http://www.sciencedirect.com/
Springer Link	http://link.springer.com/
Wiley Online Library	http://onlinelibrary.wiley.com/

Fonte: Elaborado pelo autor.

3.1.3 Critérios de inclusão e exclusão de estudos

Os critérios definidos nesta Seção foram usados para incluir e excluir estudos resultantes dos motores de busca obtidas através da *string* de busca. Portanto, esta seção lista os critérios respectivos à inclusão de estudos que fazem parte da lista de estudos primários. Esta seção também lista os critérios para remover os estudos que estão fora do escopo deste mapeamento sistemático.

A lista, a seguir, define os Critérios de Inclusão (IC) que foram utilizados para manter estudos durante o processo de seleção. Para serem mantidos na lista, os estudos precisavam atender aos seguintes requisitos:

- **IC1**: artigos, livros, capítulos de livros focados em abordagens ou algoritmos para medir a carga cognitiva do desenvolvedor;
- **IC2**: publicados ou divulgados na língua inglesa;
- **IC3**: publicados ou disponibilizados em revistas científicas como jornais, livros, conferências e *workshops*;
- **IC4**: publicados até Maio de 2020, sem uma data mínima definida.

Além disso, foram utilizados os seguintes Critérios de Exclusão (EC) que foram utilizados para remover trabalhos do processo de seleção. Os critérios de exclusão são os seguintes:

- **EC1:** o título, resumo ou qualquer parte do estudo não está relacionado com o tópico de pesquisa, mas eles são lexicalmente associados com os termos da *string* de busca;

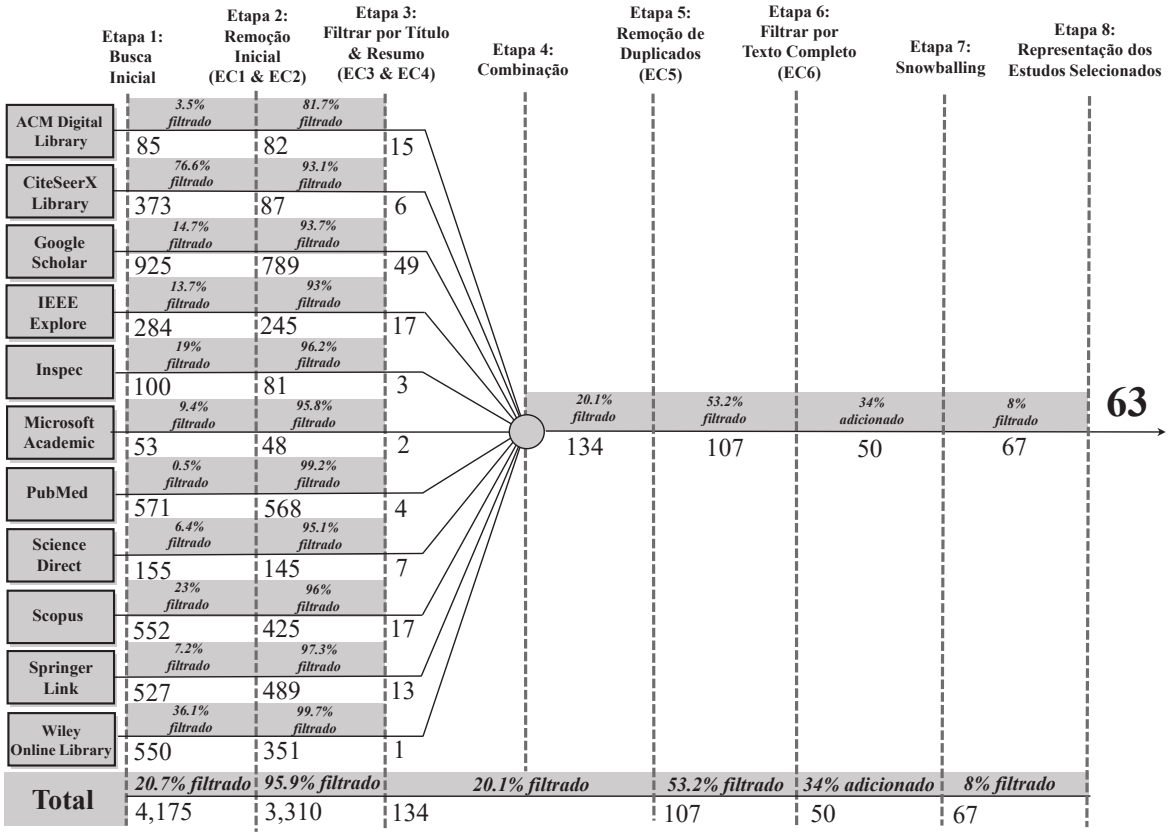
- **EC2:** o estudo não foi publicado na língua inglesa;
- **EC3:** definições de patentes, materiais não revisados por pares, estudos secundários, e descrições de apresentações;
- **EC4:** o resumo não representa um estudo no contexto de carga cognitiva em engenharia de software;
- **EC5:** o estudo é duplicado; e
- **EC6:** o estudo não aborda problemas relacionadas as questões de pesquisa definidas em relação as medidas de carga cognitiva em tarefas relacionadas a engenharia de software.

3.1.4 Processos de seleção de estudos primários

Esta Seção apresenta o processo para filtrar os estudos publicados, as quais são relacionadas as questões de pesquisas definidas na Seção 3.1.1. Este processo de filtragem consiste em oito etapas e é baseado em diretrizes bem estabelecidas pela comunidade científica (KITCHENHAM; BUDGEN; PEARL BRERETON, 2011; PETERSEN; VAKKALANKA; KUZNIARZ, 2015; KITCHENHAM; CHARTERS, 2007). Estudos de mapeamento sistemáticos recentes (GONCALES et al., 2019) demonstraram efetividade e utilidade deste processo adotado. O processo filtra gradualmente estudos potencialmente relevantes com o objetivo de obter uma amostragem de estudos representativos. A Figura 5 apresenta uma visão geral do processo de filtragem, juntamente aos resultados obtidos a cada etapa realizada. Este estudo obteve uma lista de estudos representativos da literatura após analisar um conjunto inicial de 4,175 estudos durante todo o processo de seleção, onde esse processo resultou em 63 estudos primários, todos listados no Apêndice B. Cada etapa do processo de filtragem foi executada da seguinte forma:

- **Etapa 1: busca Inicial.** O objetivo desta etapa é obter um conjunto inicial de estudos a partir dos motores de busca através da utilização da *string* definida na Seção 3.1.2.1. Ao total, foram encontrados 4,175 estudos a partir dos onze motores de busca utilizados nesta pesquisa.
- **Etapa 2: remoção inicial.** Esta etapa tem objetivo de remover resultados irrelevantes a partir dos estudos obtidos da busca inicial. Nesta etapa foram aplicados os critérios de exclusão (EC) EC1 e EC2. Deste modo, foram removidos documentos que eram chamadas para edições especiais em periódicos, chamadas de estudos para conferências, transcrições de palestras, relatórios iniciais de pesquisa, especificações de patentes, e estudos que não passaram por um processo de revisão por pares. Nesta etapa foram removidos 865 estudos, i.e., 20.7% dos estudos foram filtrados, e desta forma restaram 3,310 estudos para a próxima etapa.

Figura 5: Resultados de cada etapa do processo de filtragem.



Fonte: Elaborado pelo autor.

- **Etapa 3: filtrar por título e resumo.** O objetivo desta etapa é filtrar estudos considerando o título e o resumo. Ao aplicar o critério EC3 nesta etapa, foram removidos estudos que não continham relação com a temática desta pesquisa e as questões de pesquisa definidas neste estudo. Deste modo, 95.2% dos estudos foram filtrados, e deste modo 134 estudos restaram para serem analisados na próxima etapa.

- **Etapa 4: combinação.** Esta etapa tem o objetivo de acomodar a seleção de estudos obtidas na etapa anterior em um diretório único. Nesta etapa não foi aplicado nenhum critério de exclusão.

- **Etapa 5: remoção de duplicados.** Esta etapa tem o objetivo de garantir que os estudos sejam distintos, desde que é possível obter na amostra os mesmos estudos a partir de vários motores de busca. Foi aplicado o critério de exclusão EC5 para remover os estudos duplicados. Foram removidos 20.1% dos estudos (27 de 134) nesta etapa, restando 107 estudos para o próximo passo.

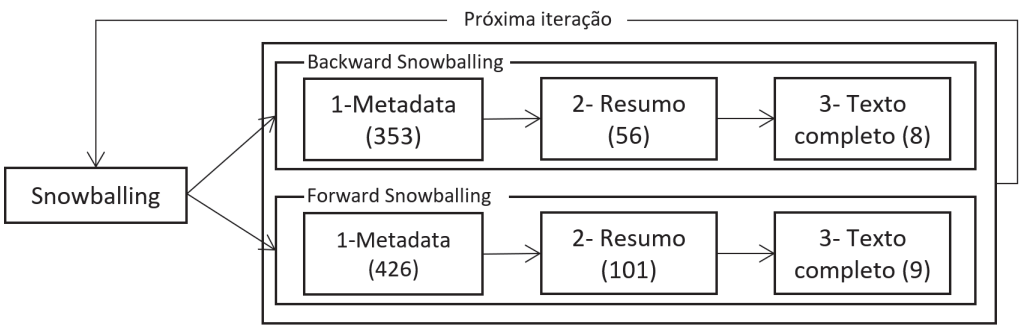
- **Etapa 6: filtragem dos estudos a partir da leitura completa do texto.** O objetivo desta etapa é filtrar estudos considerando o texto completo. Desta forma foi aplicado o critério de exclusão EC6 nos 107 estudos e então 57 desses estudos foram removidos as

quais o conteúdo não tinham relação com este estudo de mapeamento sistemático, e então resultando num conjunto de 50 estudos para a próxima etapa.

- **Etapa 7: *snowballing*.** Esta etapa tem o objetivo de expandir a abrangência do processo de seleção através da combinação da lista de estudos obtidos através da *string* de busca, com a estratégia de *snowballing* (KITCHENHAM; BUDGEN; PEARL BRERETON, 2011; PETERSEN; VAKKALANKA; KUZNIARZ, 2015). Desta forma, os artigos selecionados na última etapa foram utilizados como um conjunto inicial para interagir sobre as referências continas nesses estudos. Este é um processo conhecido como "*snowballing*" (WOHLIN et al., 2012). O processo de *snowballing* consiste em iterar sobre as referências em duas direções, i.e., *backward* (referências que estão listadas dentro do estudo) e *forward* (referências externas que citam o estudo). Em cada interação, os estudos foram filtrados através de metadados (título e local de publicação) aplicando os critérios de exclusão EC1 e EC2, resumo (EC4), e leitura do texto completo (EC6). Os resultados desse processo são apresentados na Figura 6. Através do *snowballing*, foram encontrados 17 novos estudos. Estes artigos então foram inclusos no processo de seleção, resultando em 67 estudos candidatos para formarem a lista final.

- **Etapa 8: seleção de estudos representativos.** Esta etapa tem o objetivo de realizar uma leitura adicional e verificar se no processo de seleção não continha versões diferentes do mesmo estudo. Examinando os 67 estudos que permaneceram na última etapa, foram identificados quatro estudos que representavam a análise do mesmo conjunto de dados. As versões anteriores destes estudos foram removidas e a versão mais recente foi mantida, resultado em um conjunto final de 63 estudos primários, os quais estão listados no Apêndice B.

Figura 6: Os resultados do processo de *snowballing* em relação à cada etapa.



Fonte: Elaborado pelo autor.

3.1.5 Extração de dados

Esta Seção descreve a estratégia de extração de dados aplicada para responder às questões de pesquisa definidas na Seção 3.1.1. A Tabela 4 apresenta os dados a serem extraídos nos estudos selecionados relacionados a cada questão de pesquisa.

Tabela 4: Dados que foram extraídos dos estudos selecionados.

Dados	**Descrição**	**Questão de Pesquisa**
Carga Cognitiva	A definição de carga cognitiva na engenharia de software tais como o esforço mental, ou Teoria da Carga Cognitiva.	QP1
Sensores	Os sensores as quais os estudos utilizaram para monitorar e coletar dados relacionados a carga cognitiva, tais como o EEG, e fMRI.	QP2
Métricas	A métrica utilizada para quantificar ou indicar a carga cognitiva gasta pelos participantes, tais como o *Event-Related Desynchronization* (ERD), ou tamanho da pupila	QP3
Técnica de *Machine Learning*	As técnicas de *machine learning* que os estudos primários utilizaram junto com medidas da carga cognitiva, tais como *Support Vector Machine* (SVM), e *Neural Networks*.	QP4
Propósito	Existem vários propósitos nos quais a carga cognitiva pode ser mensurada, tais como compreensão de código e dificuldade de tarefas.	QP5
Tarefas	Existem várias tarefas na engenharia de software nos quais os pesquisadores podem medir a carga cognitiva, tais como tarefas de programação ou compreensão de código.	QP6
Artefatos	Os artefatos nos quais desenvolvedores ou participantes manipularam durante as medidas de carga cognitiva, por exemplo, código fonte.	QP7
Quantidade de Participantes	A quantidade de participantes os quais pesquisadores recrutaram para participar do estudo, o qual foi coletado em intervalos de 10 participantes, por exemplo, 10-20, ou 20-30 participantes.	QP8
Métodos de Pesquisa	O método de pesquisa do estudo, por exemplo, experimento controlado, ou estudo de avaliação	QP9
Tipo de estudo	O tipo de estudo, por exemplo, conferência, periódico, *workshop*, ou capítulo.	QP10
Local de publicação	Extrair o nome do local de publicação nos quais os autores publicaram ou divulgaram o estudo primário.	QP10
Ano	Obter o ano em que o estudo foi publicado.	QP10

Fonte: Elaborado pelo autor.

3.1.6 Síntese de dados

Esta Seção apresenta o método utilizado para sintetizar os dados que foram extraídos a partir dos estudos selecionados. A síntese de dados consiste em resumir os dados obtidos para sumarizar os dados deste estudo, especificamente, para cada questão de pesquisa descrita na seção 3.1.1. Conduzir uma meta-análise nos estudos selecionados é uma tarefa desafiadora devido à falta de homogeneidade dos estudos. Para isso, o estudo confiou em análises de frequência, e de *Grounded Theory* para sintetizar as respostas das questões de pesquisa.

Foram adotadas técnicas de codificação fornecidas pelo *Grounded Theory* para codificar os termos que foram utilizados neste estudo. Esta teoria é geralmente composta por três passos para codificar os dados, incluindo codificação aberta, codificação seletiva, e codificação teórica. A codificação aberta consiste em descrever o processo e comportamentos de modo sucinto a partir dos textos baseado na percepção do autor. Desta forma, a etapa de codificação sele-

tiva identifica categorias chave para estas descrições, o qual evidencia a representatividade dos dados. Não foi seguida neste estudo uma abordagem completa do *Grounded Theory*, pois uma nova teoria não foi formada a partir dos dados coletados, deste modo não foi utilizada a etapa de codificação teórica. Além disso, este trabalho definiu questões de pesquisa antes da aplicação do *Grounded theory* o que delimitou a análise neste estudo. A análise dos dados consiste em um processo em que o autor primeiro codifica e depois categoriza os dados (STOL; RALPH; FITZGERALD, 2016). A etapa de codificar refere-se em extrair significados gerais a partir dos dados, para tornar este dado viável para ser rotulado e desta forma consequentemente classificado. A etapa de categorização contrasta os dados classificados na etapa anterior para ser contraposta entre as variações entre as diferentes percepções do autor, e de um assistente que auxiliou na classificação. Baseado nessas etapas, logo após ler os estudos classificados, as percepções do autor com o auxiliar foram anotadas em cada questão de pesquisa em um pequeno memorando. Desta forma, essas anotações foram sumarizadas e organizadas de acordo com o esquema de classificação apresentada na Figura 9 na Seção 3.2.11.

A análise dos resultados a partir da extração de dados foi realizada através da utilização planilhas. Foi utilizado o *MS Excel* para sumarizar os resultados baseados nos dados extraídos. A análise de frequência dos dados foi realizada em todas as questões de pesquisa. Entre as questões de pesquisa QP1 a QP9, foram utilizadas tabelas para sumarizar a frequência e a distribuição dos artigos selecionados para cada resposta. Deste modo, alguns estudos selecionados poderiam ser atribuídos a mais de um tipo de sensor (QP2), ou técnicas de *machine learning* (RQ4), e, portanto, esses estudos foram classificados como multimodais, e multi-algoritmos respectivamente. Desta forma, o objetivo foi fornecer respostas detalhadas para estes casos, e.g., descrever cada sensor, e cada técnica de *machine learning* utilizada nestes estudos. Na QP7, apesar de conter uma tabela contendo a classificação dos estudos e a frequência dos artefatos de software, foram descritas e detalhadas as linguagens de programação utilizadas em relação aos estudos que adotaram artefatos de código fonte. Na QP10, uma figura apresenta o local de publicação, ano, e a quantidade de estudos produzidos por ano. Desta forma, na seção de discussão, um gráfico de bolhas apresenta uma visão geral do cruzamento entre as contribuições dos estudos com o resultado de duas questões de pesquisa: propósitos (QP5) e métodos de pesquisa (QP8). A partir desta figura, foram derivados futuros desafios em relação a carga cognitiva na engenharia de software.

3.2 Resultados

Esta Seção apresenta os resultados das questões de pesquisa apresentadas na Tabela 1. Os dados relativos às questões de pesquisa formuladas serão apresentados a seguir.

3.2.1 QP1: como a carga cognitiva foi definida nos estudos?

A questão de pesquisa 1 investiga como os estudos primários definiram a carga cognitiva na engenharia de software. A carga cognitiva é um termo em que pesquisados em engenharia de software utilizam de forma não padronizada. Deste modo, essa seção classifica as definições de carga cognitiva utilizadas nos estudos de engenharia de software. Tabela 5 apresenta as classificações dos estudos primários em relação a definição de carga cognitiva utilizada.

Tabela 5: Classificação das definições de carga cognitiva utilizadas nos estudos primários.

Definições	**#**	**%**	**Estudos Primários**
Esforço mental	20	32%	[S01] [S06] [S02] [S05] [S11] [S15] [S20] [S22] [S25] [S29] [S31] [S34] [S38] [S46] [S47] [S56] [S59] [S60] [S61] [S63]
Performance cognitiva	6	10%	[S07] [S09] [S12] [S24] [S55] [S58]
Teoria da carga cognitiva	4	6%	[S04] [S13] [S14] [S33] [S36]
Carga de trabalho mental	4	6%	[S48] [S50] [S51] [S52]
Carga cognitiva	2	3%	[S03] [S16]
Não possui definição	26	41%	[S08] [S10] [S17] [S18] [S19] [S21] [S23] [S26] [S27] [S28] [S30] [S32] [S35] [S37] [S39] [S40] [S41] [S42] [S43] [S44] [S45] [S49] [S53] [S54] [S57] [S62]
Total	63	100%	

Fonte: Elaborado pelo autor.

Uma parte considerável dos estudos primários (32%, 20/63) define a carga cognitiva como um sinônimo de esforço mental que desenvolvedores de software utilizam ao realizar tarefas. Esta definição não tem relação com a Teoria da Carga Cognitiva. Nesta definição, os indicadores de carga cognitiva oscilam assim que os usuários aplicam mais esforço mental ao resolver uma tarefa. Estes estudos não apresentam uma preferência por um sensor específico ou métrica para mensurar a carga cognitiva.

Alguns estudos primários (10%, 6/63) relacionaram a carga cognitiva ao termo performance cognitiva. O campo de pesquisa de neurociência cognitiva utiliza este termo para se referir a medir quão bem desenvolvedores de software estão processando tarefas cognitivas (KLIMESCH, 1999) em termos da iteração entre a memória curta e a memória de longo termo. Pesquisadores mediram a performance cognitiva baseado na frequência de EEG, principalmente utilizando as ondas Alfa e Theta.

Um grupo de estudos primários (6%, 4/63) define o termo carga cognitiva a partir da Teoria da Carga Cognitiva (SWELLER, 2010). Esta teoria define três tipos de carga cognitiva: intrínseca, extrínseca e pertinente. Uma tarefa causa uma carga intrínseca nos desenvolvedores de software devido à complexidade inerente do material. Uma tarefa impõe uma carga cognitiva extrínseca nos usuários quando o material é mal projetado. A aquisição de novos esquemas mentais impõe a carga cognitiva natural nos usuários. Tabela 6 apresenta com que profundidade estudos consideraram a carga cognitiva na engenharia de software. Crk e Kluthe 2016 [S04] e

Petrusel, Mendling, e Reijers 2017 [S13] mediram a carga cognitiva intrínseca durante os experimentos. Autores citaram que as bandas de frequência e a fixação ocular oscilaram de acordo com a complexidade do artefato. Kluthe 2016 [S04] especificou que removeu componentes que poderiam causar carga cognitiva extrínseca aos participantes. Tallon et al. 2019 [S14] afirmou que características externas do modelo de processo impôs carga cognitiva extrínseca aos participantes. Lee et al. 2017 [S33] não especificou quais tipos de carga cognitiva eles estavam mensurando, diferente de Uysal 2013 [S36] que considerou os três tipos de carga cognitiva. Porém, estudos não seguiram o *framework* proposto por SWELLER (2010), i.e., os estudos não tiveram a intenção de reduzir a carga cognitiva extrínseca para melhorar o aprendizado dos usuários.

Tabela 6: Classificação dos tipos de carga cognitiva a partir da Teoria da Carga Cognitiva.

Estudos primários	Intrínseca	Extrínseca	Natural	Não especificou
Crk. e Kluthe 2016 [S04]	✓			
Petrusel, Mendling, e Reijers 2017 [S13]	✓			
Tallon et al. 2019 [S14]	✓	✓		
Lee et al. 2017 [S33]				✓
Uysal 2013 [S36]	✓	✓	✓	

Fonte: Elaborado pelo autor.

Alguns estudos primários (6%, 4/63) se referem a carga cognitiva carga de trabalho mental. Estes estudos utilizaram o termo carga de trabalho mental para se referir a quantidade de carga que uma tarefa pode impor a memória de curto prazo dos desenvolvedores de software. Os estudos primários nessa categoria não citam estudos da Teoria da Carga Cognitiva, nem mesmo citam estudos relacionados a neurociência cognitiva (KLIMESCH, 1999). Estes estudos utilizam a evidência de que indicadores como o nível da oxigenação da hemoglobina sobe assim que a carga cognitiva aumenta.

Cerca de 3% (2/63) dos estudos primários definiram a carga cognitiva como a quantidade de carga que uma tarefa impõe especificamente a capacidade da memória de trabalho dos usuários (PAAS; VAN MERRIËNBOER, 1994). Os estudos nessa categoria têm a característica de principalmente utilizar medidas relacionadas aos movimentos oculares.

A maioria dos estudos primários utilizaram o termo carga cognitiva, mas sem defini-lo e nem mesmo relacionar a uma literatura específica. Foram encontradas três maneiras de medir a carga cognitiva nesta categoria. A primeira maneira ([S26] [S40] [S43] [S45] [S54] [S57]) mediu o nível de oxigenação sanguínea no cérebro para identificar áreas cerebrais atividades durante a leitura de códigos fonte. O segundo padrão foram estudos primários ([S19] [S32] [S39]) mensurar a carga cognitiva com o *blood volume pulse*, taxa de batimentos cardíacos, frequência das bandas de EEG, e atividade eletrodérmica para indicar quando desenvolvedores não devem ser interrompidos. O terceiro e último padrão de estudos primários ([S41] [S44] [S62]) mediram as variações de todas as bandas de frequências do EEG para relacionar ao

esforço mental dos desenvolvedores de software durante as tarefas de programação. Por fim, a concentração de estudos primários no grupo que não definiu a carga cognitiva na Tabela 5 sugerem que os estudos que medem a carga cognitiva na engenharia de software tendem a referenciar o termo de forma inadequada.

3.2.2 QP2: quais são os tipos de sensores adotados para capturar dados relacionados a carga cognitiva dos desenvolvedores?

A questão de pesquisa 2 busca entender quais sensores pesquisadores utilizaram para medir a carga cognitiva dos desenvolvedores. Esta informação pode auxiliar outros pesquisadores e engenheiros de software a escolher quais sensores utilizar nos seus futuros estudos ou aplicações. Tabela 7 apresenta a classificação das tecnologias utilizadas para medir a carga cognitiva dos desenvolvedores.

Tabela 7: Classificação de sensores para medir a carga cognitiva dos desenvolvedores.

Sensores	**#**	**%**	**Estudos Primários**
Multimodal	27	43%	[S03][S05][S06][S07][S08][S10][S12][S15][S19][S20][S25][S29] [S30][S31][S32][S33][S34][S35][S37][S38][S39][S40][S46][S56] [S59][S61][S63]
EEG	13	22%	[S04][S09][S17][S23][S24][S41][S44][S51][S52][S55][S58][S60] [S62]
Eye-Tracker	10	16%	[S02][S11][S13][S14][S18][S21][S27][S28][S42][S47]
fMRI	8	13%	[S01][S22][S26][S43][S45][S49][S54][S57]
fNIRS	3	5%	[S36][S48][S50]
EMG	1	2%	[S53]
Não utiliza	1	2%	[S16]
Total	63	100%	

Fonte: Elaborado pelo autor.

A Tabela 7 apresenta que uma grande quantidade dos estudos primários (43%, 27/63) preferiu utilizar sensores multimodais para capturar dados relacionados a carga cognitiva, i.e., aplicar múltiplos sensores.

Estudos primários tendem a adotar sensores multimodais devido à duas razões principais. Primeiro, existe uma vasta gama de sensores que podem capturar dados relacionados a carga cognitiva a baixo custo, e.g., EEG, sensor de pele, e sensor de respiração. Segundo, a necessidade de fortalecer a correlação de resultados com o propósito do estudo. Por exemplo, alguns estudos combinaram sensores com o objetivo de combinar características temporais e espaciais. Outros estudos focaram em coletar dados a partir de diferentes partes do corpo.

A Tabela 7 também apresenta que estudos primários pouco utilizaram sensores de modo individual para medir a carga cognitiva. Estudos em engenharia de software produziram pouco conhecimento empírico em relação a cada sensor para capturar dados relacionados a carga cognitiva. Especificamente, a lista na Tabela 7 não contém nenhuma pesquisa focada em adotar sensores eletrodérmicos, batimento cardíaco, e de taxa de respiração de modo avulso para mensurar a carga cognitiva de desenvolvedores de software. Ao invés disso, estudos primários adotaram um conjunto de sensores como mostra a Tabela 8. Pesquisadores precisam explorar e avaliar com mais frequência sensores de modo individual para viabilizar a produção de sistemas de baixo custo para monitorar a carga cognitiva dos desenvolvedores.

Cerca de 22% (13/63) dos estudos primários na engenharia de software explorou o sensor EEG para medir a carga cognitiva. Este é um número baixo pois pesquisadores no campo de pesquisa sobre Interfaces Cérebro-Computador utilizam o EEG de forma constante nas pesquisas laboratoriais (LOTTE et al., 2018). Este sensor é relevante porque a área de eletrofisiologia cognitiva relaciona as leituras de EEG diretamente ao nível de interação da população neural (COHEN, 2017). Porém, pesquisadores talvez tenham prestado pouca atenção a esse sensor devido à existência de dificuldades para implantá-los no escalpo dos desenvolvedores de software, e é um sensor também sensível a ruídos externos, tais como interferências de lâmpadas e de smartphones (RADEVSKI; HATA; MATSUMOTO, 2015). Estudos também pouco focaram em utilizar sensores *Eye-Tracker* (16%,10/63). O sensor *Eye-Tracker* pode rastrear movimentos oculares na localização do artefato de software nos quais os desenvolvedores de software aplicam a carga cognitiva. A pouca utilização do *Eye-Tracker* contribui para a falta de conhecimento sobre como a carga cognitiva relacionado a unidades menores de artefatos de software, tais como uma linha específica do código fonte.

Além disso, 13% (8/63) dos estudos primários adotaram o fMRI, e cerca de 5% (3/63) adotaram o fNIRS. A quantidade de estudos que adotaram o fMRI é também baixa devido ao elevado custo do equipamento. O fMRI também necessita de um grande espaço para o equipamento. Uma alternativa a estes problemas e limitações é o sensor de fNIRS, o qual fornece informação similar comparado ao fMRI, mas com menor precisão espacial. O sensor de fNIRS não alcança o monitoramento da atividade cerebral nas regiões mais profundas, com a qual é possível com o fMRI. Mesmo com os problemas de custo e mobilidade resolvidos, apenas três estudos utilizaram o fNIRS. Sensores que possuem precisão espacial dependem da resposta do fluxo sanguíneo no cérebro e como esse fluxo é demorado, consequentemente ocorre um atraso para capturar essa dinâmica. Deste modo, pesquisadores geralmente preferem dispositivos alta precisão temporal.

Apenas 2% (1/63) dos estudos adotaram um sensor EMG para investigar a carga cognitiva. Este sensor captura os sinais elétricos emitidos pelos movimentos musculares da face. Os estudos negligenciaram a ideia de utilizar alguns padrões dos músculos faciais para indicar a carga cognitiva dos desenvolvedores. Músculos faciais também podem refletir palavras que desenvolvedores estão pensando, um processo chamado de subvocalização.

Tabela 8: Classificação dos estudos primários que utilizaram múltiplos sensores.

Estudos Primários	**Conjunto de sensores combinados**											
	EEG	***Eye-tracker***	**GSR**	**EDA**	**EMG**	**fNIRS**	**fMRI**	**ECG**	**RR**	**HR**	**PPG**	**Movimento**
Nargess Nourbakhsh 2013 [S03]		✓	✓									
Sarah Fakhoury 2018 [S05]		✓				✓						
Thomas Fritz 2016 [S06]	✓	✓		✓						✓		
Magdalena Borys 2017 [S07]	✓	✓										
Sebastian C. Müller 2015 [S08]	✓	✓		✓						✓		
Daniela Girardi 2017 [S10]	✓	✓		✓							✓	
Randall Minas 2017 [S12]	✓			✓	✓							
Norman Peitek 2019 [S15]		✓					✓					
Manuela Züger 2015 [S19]	✓	✓		✓								
Norman Peitek 2018 [S20]		✓					✓					
Davide Fucci 2019 [S25]	✓			✓						✓		
Sebastian C. Müller 2016 [S29]				✓					✓	✓		
Sebastian C. Müller 2015 [S30]	✓	✓		✓								
Norman Peitek 2018 [S31]		✓					✓					
Manuela Züger 2018 [S32]	✓			✓						✓		
Seolhwa Lee 2017 [S33]	✓	✓										
Ricardo Couceiro 2019 [S34]	✓	✓		✓				✓				
Toyomi Ishida 2019 [S35]	✓	✓										
Thomas Fritz 2014 [S37]	✓	✓		✓								
Florian Schaule 2018 [S38]			✓							✓		
Manuela Züger 2018 [S39]								✓		✓		✓
Yu Huang 2019 [S40]						✓	✓					
Norman Peitek 2018 [S46]		✓					✓					
Sarah Fakhoury 2020 [S56]		✓				✓						
Ricardo Couceiro 2019 [S59]		✓		✓				✓				
Ricardo Couceiro 2019 [S61]		✓								✓		
Devjeet Roy 2020 [S63]		✓					✓					

Legend:
EEG: Eletroencefalograma, GSR: Galvanic Skin Response, EDA: Electro Dermal Activity, EMG: Eletromiografia,
fNIRS: Functional near-infrared spectroscopy, fMRI: Functional Magnetic Resonance Imaging,
ECG: Eletrocardiografia, RR: Respiration Rate, HR: Heart Rate, PPG: Pletismógrafo

Fonte: Elaborado pelo autor.

A Tabela 8 apresenta os estudos que utilizaram sensores multimodais. Na direção horizontal, a Tabela 8 apresenta quais sensores os estudos primários utilizaram para medir a carga cognitiva. Na direção vertical, a Tabela 8 apresentam quais estudos primários adotaram um tipo específico de sensor.

A Tabela 8 evidencia que os estudos primários combinaram os sensores de EEG, *Eye-Tracker*, e EDA com mais frequência. A combinação desses sensores viabiliza a análise das ondas cerebrais e também realizar o rastreamento de onde o desenvolvedor está observando no artefato de software. Os estudos primários também utilizaram os sensores de taxa de batimento cardíaco com frequência, os quais também possuem correlação com a carga cognitiva. Os espaços em branco na Tabela 8 evidencia que poucos estudos combinaram a informação a partir do fMRI ou fNIRS com um sensor que possua maior precisão temporal, tais como, o *Eye-Tracker*, ou o EEG. A falta dessa combinação evidencia uma lacuna na pesquisa em engenharia de software em relacionar dinâmicas cerebrais que envolvem a carga cognitiva em locais específicos de artefatos de software com mais detalhes. Para isso, pesquisadores em engenharia de software poderiam explorar um campo de pesquisa da neurociência dedicado a análise conjunto de dados do EEG e fMRI chamado de EEG-fMRI (DEBENER et al., 2005). A falta de explorar a utilização de sensores GSR e ECG indicam que faltam aspectos mais precisos em relação a

batimentos cardíacos e dados da epiderme.

A Tabela 8 também apresenta os estudos primários raramente adotaram o pletismógrafo e sensores de movimento. Mas a falta da exploração desses sensores não tem impacto na relevância das pesquisas. Sensores de batimento cardíaco podem capturar dados de respiração. Os sensores de EEG e *Eye-Tracker* podem capturar movimentos de interesse aos pesquisadores, i.e., piscar e movimentos sacádico dos olhos.

3.2.3 QP3: quais métricas foram usadas para medir a carga cognitiva dos desenvolvedores?

A questão de pesquisa 3 classifica quais métricas os estudos primários utilizaram para medir a carga cognitiva dos desenvolvedores. O objetivo desta questão é fornecer uma visão geral dos indicadores que os pesquisadores utilizaram para medir a carga cognitiva dos desenvolvedores. Tabela 9 apresenta a classificação dos estudos primários em relação as medidas da carga cognitiva.

Tabela 9: Classificação de estudos primários baseados nas métricas relacionadas à carga cognitiva.

Métricas	**#**	**%**	**Estudos primários**
Multimodais	39	62%	[S02][S03][S05][S06][S07][S08][S10][S11][S13][S14] [S17][S19][S20][S21][S23][S24][S25][S27][S28][S29] [S30][S31][S32][S33][S34][S35][S37][S38][S39][S40] [S42][S45][S46][S48][S56][S57][S58][S59][S61]
Níveis de Oxigenação Sanguínea	8	13%	[S01][S22][S26][S36][S43][S49][S50][S54]
Bandas de frequência	6	10%	[S41][S51][S52][S53][S60][S62]
Event-Related Desincronization (ERD)	4	7%	[S04][S12][S44][S55]
Fixação ocular	2	3%	[S18][S47]
Individual Alpha Frequency (IAF)	1	2%	[S09]
Não especificou	3	5%	[S15][S16][S63]
Total	63	100%	

Fonte: Elaborado pelo autor.

A maioria dos estudos primários (62%, 39/63) utiliza métricas multimodais para medir a carga cognitiva dos desenvolvedores. Estudos primários que utilizaram ou aplicaram mais que um indicador foram classificados como “multimodal”. Pesquisadores no campo de sistemas de reconhecimento (BLASCO et al., 2016; BABLANI et al., 2019) também utilizam métricas multimodais com o propósito de superar inconsistências durante a análise de dados. Desta forma, falhas causadas por leituras em um ou mais sensores pode ser corrigido por outra métrica. Mesmo assim, grande parte dos estudos primários (35%, 22/63) utilizaram apenas um tipo de métrica para medir a carga cognitiva dos desenvolvedores de software. Pesquisadores extraíram a maioria desses indicadores a partir da atividade cerebral, i.e., Nível de oxigenação sanguínea

(13%, 8/63). Pesquisadores geralmente extraem esses tipos de métricas a partir dos sensores de fMRI ou NIRS. Além disso, cerca de 19% (11/63) dos estudos primários adotaram métricas obtidas do eletroencefalograma, tais como bandas de frequência (10%, 6/63), *Event-Related Desynchronization* (7%, 4/63), e *Individual-Alpha Frequency* (2%, 1/63).

Pesquisadores utilizaram o Nível de Oxigenação Sanguínea para estimar a região cerebral que correlaciona com tarefas de engenharia de software (SIEGMUND et al., 2017; PEITEK et al., 2018a). O nível de oxigenação sanguínea é uma métrica consolidada para indicar áreas de ativação cerebral. Pesquisadores utilizaram o *Event-Related Desynchronization* (ERD) para mensurar o esforço mental durante a compreensão de código (CRK; KLUTHE; STEFIK, 2015; CRK; KLUTHE, 2016). Esta medida é útil para capturar a dessincronização da atividade cerebral durante eventos de compreensão. Um exemplo da aplicação das ondas de frequência, KOSTI et al. (2018) demonstraram que uma diferença estatisticamente significativa nas bandas de frequência poderia evidenciar o nível da dificuldade das tarefas de desenvolvedores de software. Em geral, a presença de métricas tais como as bandas de frequência, ERD, e IAF nos estudos primários mostram que pesquisadores tem um entendimento comum que as ondas cerebrais têm correlação com a carga cognitiva, mas uma divergência sobre qual aspecto onda cerebral deve ser considerada para mensurar a carga cognitiva. Pesquisadores da área de neurofisiologia relacionaram métricas baseadas em ondas cerebrais aos níveis de carga cognitiva dos desenvolvedores de software em atividades que exigem carga de trabalho mental tais como, jogos de xadrez.

Apenas dois estudos focaram em rastrear movimentos oculares dos desenvolvedores. A fixação ocular representa onde desenvolvedores estão observando no artefato de software. A quantidade de estudos focando exclusivamente nesta métrica implica que há pouco conhecimento em relação aos pontos de interesse dos desenvolvedores no código fonte. Contudo, pesquisadores em engenharia de software também focaram menos na fixação ocular porque eles têm preferência em combinar técnicas de *Eye-Tracking* com o EEG. Estudos que não definiram as métricas consistem em pesquisas em estágio inicial.

A Tabela 10 apresenta os trabalhos que utilizaram métricas multimodais. Na direção horizontal, a Tabela 10 apresenta as métricas que estudos primários utilizaram para medir a carga cognitiva. Na direção vertical, a Tabela 10 apresenta os estudos primários que aplicaram as métricas indicadas em cada coluna. No geral, estudos primários não tem uma preferência por uma métrica para medir a carga cognitiva. Existem apenas algumas métricas que os estudos primários adotaram com mais frequência. Os estudos primários tendem a utilizar todas as ondas de frequência (AFB) a partir do sensor de EEG (12/38). A partir do *Eye-Tracker*, estudos geralmente adotaram a métrica de "piscares de olhos" (9/38). Estudos primários também tiveram preferência para medir atividade eletrodérmica (EDA) a partir dos sensores de atividade dermal (pele). Apesar da identificação de quatro métricas a partir do sensor de fNIRS, os estudos primários focaram na utilização de duas, a HbO (Hemoglobina Desoxigenada) e o HbR (Hemoglobina Oxigenada). O Pulso de Volume Sanguíneo (BVP) obtido a partir do fMRI foi

Tabela 10: Classificação de estudos primários baseados em métricas multimodais.

Estudos Primários	EEG											*Eye-tracker*											Pele					fNIRS				fMRI			Coração						PA
	AFb	FBR	Ab	Bb	Tb	Gb	ERD	PS	AR	AL	ML	EB	EF	NF	NS	SD	FD	GPL	BN	BR	PDS	VI	AG	PSG	ST	SCR	EDA	HbT	HbO	HbR	Oxy	BOL	I	V	HR	HRV	BVP	RR	SL	TPE
Michael Zimoch 2012 [S02]														✓	✓			✓																						
Nargess Nourbakhsh 2013 [S03]																			✓	✓			✓	✓																
Sarah Fakhoury 2018 [S05]																												✓	✓	✓	✓									
Thomas Fritz 2016 [S06]	✓											✓	✓								✓						✓								✓	✓	✓	✓		
Magdalena Borys 2017 [S07]																																								
Sebastian Müller 2015 [S08]		✓	✓																							✓														
Daniela Girardi 2018 [S10]	✓											✓	✓								✓						✓								✓	✓	✓			
Razvan Petrusel 2016 [S11]														✓			✓																						
Razvan Petrusel 2017 [S13]														✓			✓																						
Miles Tallon 2019 [S14]														✓	✓																								
Seolhwa Lee 2016 [S17]			✓	✓	✓	✓																																	
Manuela Züger 2015 [S19]	✓																								✓										✓					
Mahnaz Behroozi 2018 [S21]												✓			✓							✓																		
Jefferson Seide Molléri 2019 [S23]										✓	✓																													
Aruna Duraisingam 2017 [S24]	✓	✓					✓		✓																														
Davide Fucci 2019 [S25]	✓																								✓										✓	✓				
Katja Kevic 2016 [S27]												✓	✓			✓																								
Martin Konopka 2015 [S28]													✓								✓																			
Sebastian C. Müller 2016 [S29]																									✓		✓								✓	✓		✓		
Sebastian C. Müller 2015 [S30]	✓											✓	✓								✓						✓								✓	✓	✓	✓		
Norman Peitek 2018 [S31]												✓									✓											✓								
Manuela Züger 2018 [S32]	✓											✓															✓								✓	✓			✓	
Seowlwa Lee 2017 [S33]	✓											✓									✓																		
Ricardo Couceiro 2019 [S34]	✓																																			✓	✓		
Toyomi Ishida 2019 [S35]			✓											✓																									
Thomas Fritz 2014 [S37]	✓											✓															✓												
Florian Shaule 2016 [S38]																							✓		✓										✓	✓				
Manuela Züger 2018 [S39]																																			✓	✓			✓	✓
Yu Huang 2019 [S40]																													✓	✓		✓								
Mahnaz Behroozi 2018 [S42]															✓			✓				✓																		
Benjamin Floyd 2017 [S45]																																	✓	✓						
Norman Peitek 2018 [S46]														✓	✓																✓								
Yoshiharu Ikutani 2014 [S48]																													✓	✓									
Sarah Fakhoury 2019 [S56]														✓														✓	✓	✓	✓								
Ryan Krueger 2020 [S57]																																	✓	✓						
Ramaswamy Palaniappan 2019 [S58]							✓	✓	✓																														
Ricardo Couceiro 2019 [S59]	✓																				✓					✓										✓				
Ricardo Couceiro 2019 [S61]	✓																				✓															✓				

Legenda:
AFb: todas as bandas de frequência, FBR: taxa de banda de frequência, Ab: alfa, Bb: beta, Tb: theta, Gb: gama, ERD: *Event-Related Desynchronization*, AR: taxa de assimetria, AL: nível de atenção, ML: nível de meditação,
EB: piscar de olhos, EF: fixação ocular, NF: quantidade de fixações, NS: quantidade de sacadas, PSS: tamanho da pupila, SD: duração de sacadas, NS: quantidade de sacadas, FD: duração da fixação, BN: quantidade de piscar de olhos, BR: taxa de piscar de olhos,
GPL: *Gaze Path Length*, VI: *Visual Intake*, AG: GSR acumulativa, PSG: potência espectral GSR, ST: temperatura de pele, SCR: resposta de condutância da pele, EA: atividade eletrodérmica, HbT: hemoglobina total,
HbO: hemoglobina desoxigenada, HbR: hemoglobina oxigenada, Oxy: alterações totais na oxigenação, BOL: nível da oxigenação do sangue, I: imagens, V: *voxels*, HR: taxa de batimento cardíaco, HRV: variabilidade de taxa de batimento cardíaco,
BVP: pulso de volume sanguíneo, RR: taxa de respiração, SL: nível de sono, TPE: tempo gasto em atividade física, PA: atividade física

Fonte: Elaborado pelo autor.

utilizado em três estudos primários. Os estudos primários também utilizaram a métrica HRV - variabilidade de frequência cardíaca (11/38) a partir dos sensores que capturam atividade cardíaca.

Na direção horizontal, a Tabela 10 apresenta aspectos positivos e negativos em relação as métricas que os estudos primários combinaram. O aspecto positivo é que existe uma ampla gama de métricas que podem ser obtidas de cada sensor. Desta forma, a partir de um sensor, pesquisadores possuem várias opções para mensurar a carga cognitiva. O aspecto negativo é que as lacunas indicam que os estudos primários não exploraram todas as métricas disponíveis em cada sensor. Desta forma, é evidente que os estudos primários que preferiram combinar métricas poderiam explorar e agregar mais métricas. Porém, eles aplicaram um pequeno conjunto de métricas de cada sensor. A utilização de várias métricas a partir de um mesmo sensor pode tornar a análise mais confiável de ser conduzida. Por exemplo, no *Eye-Tracker*, enquanto uma frequência reduzida nos piscares de olhos indica alta demanda da carga cognitiva, pesquisadores também podem utilizar um tamanho reduzido da pupila para confirmar esta alta demanda da tarefa.

3.2.4 QP4: quais algoritmos foram usados para classificar a carga cognitiva do desenvolvedor?

A questão de pesquisa 5 tem o objetivo de classificar diferentes técnicas de *machine learning* que os estudos primários utilizaram. A Tabela 11 apresenta uma lista das técnicas de *machine learning* utilizadas pelos estudos primários. A Tabela 11 apresenta que 30% (18/63) dos estudos primários geralmente aplicaram técnicas de classificação. Cerca de 11% (7/68) dos estudos primários avaliaram mais de um algoritmo de *machine learning* (*Multi-algoritmos*).

Tabela 11: Técnicas de *machine learning* utilizadas com métricas da carga cognitiva.

Técnicas de *Machine Learning*	**#**	**%**	**Estudos primários**
Multi-algoritmos	7	11%	[S03][S07][S21][S25][S38][S39][S58]
Naïve Bayes (NB)	6	10%	[S06][S19][S24][S30][S32][S37]
Support Vector Machine (SVM)	3	5%	[S23][S33][S34]
Decision Tree (DT)	1	2%	[S08]
Random Forest (RF)	1	2%	[S29]
Não utiliza	41	65%	[S01][S02][S04][S05][S09][S11][S12][S13][S14][S16] [S17][S18][S20][S22][S26][S27][S31][S35][S36][S40] [S41][S42][S43][S44][S45][S46][S47][S48][S49][S50] [S51][S52][S53][S54][S55][S56][S57][S59][S60][S61] [S62]
Não especifica	4	6%	[S10][S15][S28][S63]
Total	63	100%	

Fonte: Elaborado pelo autor.

A Tabela 11 mostra que as técnicas *Naïve Bayes* (NB) e *Support Vector Machine* (SVM) foram as mais utilizadas entre os estudos primários que utilizaram apenas uma técnica (15%,

9/63). Os estudos que adotaram *Naïve Bayes* conduziram comparações com outras técnicas de *machine learning* tais como a técnica *Support Vector Machine* (SVM) e a técnica *Decision Tree* (DT) antes de utilizar a técnica de maneira definitiva. Pesquisadores optaram por utilizar a técnica *Naïve Bayes* porque obteve uma performance superior as técnicas SVM e *Decision Tree*. Desta forma, pesquisadores preferiram reportar os resultados obtidos pela *Naïve Bayes* ao invés dos resultados obtidos pelos classificadores SVM e DT. Isto é, há de certa forma uma limitação das pesquisas sobre carga cognitiva na engenharia de software. A adoção da técnica *Naïve Bayes* e *Support Vector Machine* são coerentes com os resultados presentes em GUI et al. (2019); BLASCO et al. (2016).

Uma minoria dos estudos primários adotou outras técnicas de classificação tais como a técnica *Decision Tree* (2%, 1/63), e *Random Forest Learners* (2%, 1/63). *Random Forest* é uma técnica que os estudos primários negligenciaram pois tem a característica de ser precisa mesmo com um conjunto de dados pequeno, o conhecido curso de dimensionalidade. Poucos estudos primários (6%, 4/63) não especificaram a técnica de *machine learning* porque esses estudos são propostas de solução que pretendem utilizar técnicas de *machine learning*, porém não especificaram quais técnicas pretendiam utilizar.

Outro resultado importante é que as técnicas de *machine learning* tais como o *K-Nearest Neighbor* e *Neural Networks*, não estão relacionadas na Tabela 11 porque essas técnicas são suscetíveis a *overfitting* caso sejam treinadas com dados psicofisiológicos. Pesquisas na área de Interfaces Cérebro-Computador (BCI) apontam que essas técnicas são suscetíveis a problemas de *overfitting* principalmente com dados de EEG (LOTTE et al., 2018). Devido a essas limitações, pesquisadores adotaram as técnicas KNN e *Neural Networks* em conjunto com as técnicas SVM e Naïve Bayes para comparar se a eficácia dessas técnicas é coerente.

Tabela 12: Lista de estudos primários que aplicaram mais de uma técnica de *machine learning*.

Estudos Primários	**Conjunto de técnicas de *machine learning*.**										
	SVM	**NB**	**DT**	**LDA**	**LR**	**KNN**	**NN**	**RF**	**GB**	**QDA**	**EC**
Nargess Nourbakhsh 2013 [S03]	✓	✓									
Magdalena Borys 2017 [S07]	✓		✓	✓	✓	✓					✓
Mahnaz Behroozi 2018 [S21]	✓	✓	✓		✓	✓	✓	✓			
Davide Fucci 2019 [S25]	✓	✓				✓	✓	✓			
Florian Schaule 2018 [S38]	✓	✓						✓			
Manuela Züger [S39]	✓	✓					✓	✓	✓		
Ramaswamy Palaniappan [S58]	✓	✓	✓			✓	✓	✓		✓	

Legenda:

SVM: Support Vector Machine, NB: Naïve Bayes, DT: Decision Tree, LDA: Linear Discriminant Analysis, LR: Logistic Regression, KNN: K-Nearest Neighbor, EC: Ensemble Classifier, RF: Random Forest, GB: Gradient Boosting, QDA: Quadratic Discriminant Analysis

Fonte: Elaborado pelo autor.

Na direção vertical, a Tabela 12 apresenta quais estudos primários contém uma técnica de *machine learning* específica. Deste modo, as técnicas *Support Vector Machine*, *Naïve Bayes*,

e *Random Forest* são as técnicas mais combinadas entre os estudos primários. Como anteriormente mencionado, SVM e RF são técnicas robustas para um conjunto de treinamento pequeno, o qual justifica a preferência dos estudos primários por estas técnicas. A Tabela 12 apresenta que apenas alguns estudos avaliaram a efetividade das técnicas *Decision Tree*, *Linear Discriminant Analysis*, *Linear Regression*, *Gradient Boosting*, e *Quadratic Discriminant Analysis*. Estas técnicas de *machine learning* tem um potencial limitado para trabalhar de maneira efetiva com dados psicofisiológicos. Porém, modificações e aprimoramentos nessas técnicas podem torná-las efetivas. Por exemplo, não foi notado nos estudos primários a aplicação de técnicas de *machine learning* adaptativas. Técnicas de *machine learning* adaptativas viabilizam a atualização de parâmetros de classificação de acordo com a adição incremental de novos conjuntos de dados. Outro exemplo é a versão modificada da técnica LDA, i.e., a sLDA a qual é robusta para um conjunto de dados menor, contudo estudos primários não exploraram essa possiblidade (LOTTE et al., 2018).

Na direção horizontal, a Tabela 12 também apresenta que apenas um estudo primário realizou a combinação dos resultados de múltiplos classificadores. Esta prática é conhecida como *ensemble classifiers* é relevante para esta área de pesquisa. Pesquisadores perderam a oportunidade de explorar o potencial dos *ensemble classifiers* junto da avaliação da efetividade individual dos classificadores.

Características dos conjuntos de dados utilizados para treinar técnicas de *machine learning* nos estudos primários: A Tabela 13 apresenta a descrição dos conjuntos de dados utilizados nos estudos primários, tais como o propósito para a utilização dos conjuntos de dados, o tipo de dado utilizado para treinar as técnicas de *machine learning*, a quantidade de participantes, a quantidade de tarefas e sua duração, os filtros aplicados nos dados coletados, e o formato dos dados. Estudos primários geraram conjunto de dados para aplicações chave na engenharia de software tais como a classificação da carga cognitiva, dificuldade de tarefas, classificação de emoções, interrupção do trabalho, e a detecção de problemas de qualidade de código.

Estes conjuntos de dados geralmente contêm dados de uma grande quantidade de participantes fazendo a utilização de sensores multimodais tais como o EDA, EEG, e HRV (FUCCI et al., 2019)[S25]. MÜLLER; FRITZ (2016) [S34] produziram um conjunto de dados que contém HR, HRV, RR e EDA. MÜLLER; FRITZ (2016) coletaram dados a partir de 10 participantes para classificar a presença de problemas de qualidade de código. Apesar do baixo número de participantes, este conjunto de dados contém um tamanho considerável devido ao monitoramento contínuo durante um dia completo de trabalho. Apesar do alto valor desses conjuntos de dados, eles não estão disponíveis devido às questões de confidencialidade dos dados dos participantes.

Efetividade: reportar qual técnica de *machine learning* tem melhor efetividade é uma tarefa desafiadora por três principais motivos. Primeiro, existe uma diferença no tamanho dos conjuntos de dados utilizados para treinamento. E o segundo motivo é que pesquisadores não possuem

Tabela 13: Lista dos conjuntos de dados produzidos pelos estudos primários.

Estudo	Type	Dado	Participantes	Quantidade de Tarefas/Duração	Filtros	Formato
[S03]	Classificação da Carga Cognitiva.	Galvanic Skin Response em taxa de amostragem de 10Hz, dados oculares, i.e., quantidade de piscares de olhos registrado a 50Hz	13	8 tarefas aritméticas	Não especificou	Texto
[S07]	Classificação da Carga Cognitiva.	Sinal EEG utilizando Mitsar EEG 201 com 21 canais e taxa de amostragem de 500Hz, Dados oculares registrados a 50Hz com um ETG 2.0	20	6 tarefas aritméticas, 5,5 segundos cada, um segundo de descanso entre as tarefas.	Remoção de movimentos com PCA nos dados EEG.	Texto
[S08] [S06]	Classificação de Emoções	Ondas cerebrais EEG obtidas a partir de canal único a 512 Hz. Atividade eletrodermal (EDA), temperatura de pele, taxa de batimento cardíaco, BVP, e tamanho da pupila a partir do *Eye-Tracker* Eye Tribe.	17	Duas tarefas de programação, 30 minutos cada tarefa. Um intervalo entre as duas tarefas, que é um vídeo de dois minutos apresentando um peixe (tarefa de relaxamento).	*Low-pass*, e *high-pass* filtro *butterworth* aplicado no sinal EDA.	Texto
[S19] [S06]	Interrupção do Trabalho	Sinal EEG obtido a partir de um Neurosky registrado a 512 Hz, este conjunto de dados contém bandas de frequência, EDA, e SCL coletados a partir de uma pulseira eletrônica, medidas oculares obtidas a partir de um *Eye-Tracker*.	10	Um seção de duas horas contínuas de tarefas de programação.	Aplicação de um *notch filter* de 50Hz no sinal EEG .Filtro *butterworth* para tratar os sinais oculares,	Texto
[S21]	Níveis de Stress	Dados de piscares de olhos e sacadas obtidos a partir do SMI em uma taxa de amostragem de 60Hz	11	Duas tarefas de programação conduzidas em um quadro branco, e outra conduzida em papeis branco/Limite de tempo das tarefas não reportado.	Sem filtros aplicados	Texto
[S24]	Dificuldade de Tarefas	Sinal EEG capturado com o headset Emotiv Epoc , com 14 canais, taxa de amostragem de 128 Hz. Este conjunto de dados contém a energia de diferentes bandas, ERD, taxas de frequência, taxa de assimetria.	8	20 questões de compreensão de código, estado de relaxamento de 30 segundos entre as tarefas.	*Elliptic IIR filter*, filtro *band-pass* entre 0.5-3 Hz	Texto
[S25]	Compreensão de Programas	Sinais EEG obtidos de um canal (512 Hz), e EDA (4Hz), e BVP (64Hz) a partir do Empatica E4.	28	Três seções contendo de compreensão de código (60 sec) e seis tarefas textuais (30 segundos) compreensão de tarefas.	Filtro *Bandpass* aplicado nos sinais de EEG e dados BVP.	Texto
[S29]	Qualidade de código fonte	HR, HRV, RR e EDA obtidos a partir de uma pulseira eletrônica e um sensor torácico *Sense* core. Métricas de código fonte tais como complexidade de McCabe , e complexidade de Halstead	10	Um dia contínuo de trabalho com tarefas de programação.	filtro butterworth sinal EDA.	Texto
[S30] [S06]	Dificuldade de Tarefa	Signal EEG obtido de um Neurosky registrado a 512 Hz, este conjunto de dados contém EDA, e SCL coletados a partir de uma pulseira eletrônica, medidas oculares a partir de um *Eye-Tracker*.	15	8 pequenas tarefas de compreensão	*notch filter* de 50Hz no sinal EEG. Filtro *butterworth* para filtrar componentes oculares, filtro *Butterworth* no sinal EDA	Texto
[S32]	Interrupção de Trabalho	Sinal EEG obtido a 512 Hz, Empatica E3 para capturar sinal EDA, e temperatura da pele e BVP, HR obtido do sensor Polar H7.	33	10 desenvolvedores conduziram tarefas de compreensão por uma hora; 10 participantes conduziram tarefas de programação por duas horas; 13 participantes conduziram tarefas de programação por duas semanas. Desenvolvedores sofreram interrupções intermitentes durante todas as tarefas.	Não especificou	Texto
[S33]	Nível de Conhecimento.	Sinal EEG com 32 canais a partir do sistema iCap o qual registra em uma taxa de amostragem de 500 Mhz, dados com a posição ocular.	38	23 tarefas de compreensão com cerca de 60 segundos, tarefa de relaxamento de 60 segundos.	Não especificou	Texto
[S34]	Classificação da Carga Cognitiva.	Variabilidade de batimento cardíaco (HRV) a partir de um ECG.	26	3 tarefas de compreensão com tempo limite de 10 minutos, atividade de controle duração de 60 segundos.	Não especificou	Texto
[S37]	Dificuldade de Tarefas.	Sinal EEG registrado a 512Hz a partir de 1 canal. EDA registrado a 8Hz a partir de um sensor Affectiva Q Rastreamento ocular registrado a 300 Hz a parir do *Eye-Tracker* TX300	15	8 tarefas de compreensão.	Remoção de componente DC do EDA, filtro *low-pass* e *butterworth* no EDA. *Notch filter* de 60Hz no sinal de EEG.	Text
[S38]	Interrupção de Trabalho	GSR, Temperatura da Pele, Taxa de Batimento Cardíaco (HR), Variabilidade de Taxa de Batimento Cardíaco (HRV)	20	3 tarefas que consistem em memorizar um quadrado, e equações matemáticas, cada tarefa tem um limite de 8 minutos. Participantes foram interrompidos intermitentemente	Artefatos de movimentos musculares foram removidos dos dados.	Texto
[S39]	Interrupção do Trabalho.	dados HR e HRV a partir do sensor Polar H7, HR, atividade física, e dados de sono a partir do Fitbit Charge 2.	13	Monitoramento de atividades diárias dos participantes, os quais vestiram sensores durante todo o seu dia de trabalho.	Não especificou.	Texto
[S58]	Dificuldade de Tarefas.	Sinal de EEG a partir do Emotiv Epoc+ com 14 canais, ERD, taxa de assimetria (ASR).	9	6 tarefas de compreensão, o qual o tempo para completar variou de 35 a 210 segundos.	*Elliptic IIR filters*	Texto

Fonte: Elaborado pelo autor.

consenso sobre quais indicadores de efetividade devem ser utilizados. Por exemplo, alguns estudos primários utilizam apenas a métrica de sensitividade; outros medem a precisão e o *recall*. Terceiro, os estudos treinaram as técnicas de *machine learning* em diferentes condições, algumas técnicas adotam uma validação cruzada de 10 *Folds*, outros realizam uma avaliação com 5 *Folds*. Portanto, existe a falta de uma análise unificada dessas técnicas na literatura para analisar a efetividade dessas técnicas.

Desta forma, foi sumarizado alguns dos melhores resultados das técnicas de *machine learning* que a literatura obteve a partir do treinamento nos respectivos conjuntos de dados apresentados na Tabela 13.

- **Classificação da Carga Cognitiva:** uma técnica KNN alcançou uma acurácia de 90,4% para classificar carga cognitiva baseados em dados obtidos do *Eye-Tracker* utilizando validação cruzada de 5 *folds* [S07];

- **Emoções:** um classificador de árvore J48 treinado com taxas de frequências de banda, variabilidade de taxa de batimento cardíaco e tamanho da pupila alcançou 64,32% de precisão e 82.03% de *recall* para classificar emoções negativas e positivas treinados com o método*leave-one-out* [S08];

- **Nível de Experiência:** uma técnica SVM treinada com dados do EEG e *Eye-Tracker* alcançou uma precisão de 97,7%, um *recall* de 96,4% e *F-measure* de 97% para classificar entre duas classes de nível de conhecimento (Principiante / Especialista) em uma validação cruzada de 10 *fold* através de uma abordagem *leave-one-out* [S33];

- **Compreensão de Programas:** uma técnica *Neural Networks*, especificamente uma *multi-layer perceptron* alcançou 95% de precisão e 90% de *recall* para identificar se um participante está interagindo com uma tarefa de código fonte ou um texto simples baseados no sinal cardíaco treinado com validação cruzada *hold-out* [S25];

- **Problemas de qualidade de código:** uma técnica *Random forest learner* treinada com HR, HRV, RR, e EDA alcançou 13% de precisão e 38,6% de *recall* para identificar a presença ou não de cinco categorias de problema de qualidade de código [S29].

- **Dificuldade de Tarefas:** uma técnica *Naïve Bayes* alcançou uma precisão de 90,1%, e um *recall* de 73,4% para classificar dois níveis de dificuldade (fácil/difícil)[S24];

- **Interrupção do Trabalho:** A acurácia do classificador *Naïve Bayes* alcançou 91,5% em um estudo de laboratório, enquanto que 78,6% no ambiente de trabalho para classificar a propensão de ser interrompido ou não [S19].

3.2.5 QP5: para qual propósito a carga cognitiva dos desenvolvedores foi medida?

A questão de pesquisa 5 investiga com que propósito a área de engenharia de software mediu a carga cognitiva. Tabela 14 apresenta a classificação dos estudos primários de acordo com os propósitos em que a carga cognitiva foi medida.

A maioria dos estudos primários (38%, 24/63) se concentrou em analisar a carga cognitiva dos desenvolvedores de software para analisar compreensão de código. O foco em investigar compreensão de código evidenciado na Tabela 14 é um fato em que a literatura de *Eye-Tracking* na engenharia de software também identificou (SHARAFI; SOH; GUÉHÉNEUC, 2015; OBAIDELLAH; AL HAEK; CHENG, 2018). Compreensão de programas é uma atividade básica na engenharia de software. O principal foco desses estudos foi investigar quais áreas cerebrais estavam envolvidas na compreensão de programas. Além disso, compreensão de código é um

Tabela 14: Classificação de estudos primários baseados no propósito para medir a carga cognitiva.

Propósito	**#**	**%**	**Estudos Primários**
Compreensão de código	24	38%	[S01][S02][S05][S09][S11][S13][S14][S15][S17][S18] [S20][S22][S35][S43][S44][S45][S46][S47][S48][S52] [S54][S56][S57][S63]
Dificuldade de tarefas	12	19%	[S03][S06][S24][S26][S27][S30][S31][S37][S40][S41] [S53][S58]
Interrupção	6	10%	[S19][S23][S32][S38][S39][S55]
Classificação da Carga Cognitiva	5	8%	[S07][S34][S42][S50][S59]
Problemas de Qualidade	4	6%	[S29] [S49] [S61] [S62]
Nível de Conhecimento	3	5%	[S04][S25][S33]
Produtividade	3	5%	[S08][S16][S51]
Reconhecimento de Emoções	2	3%	[S10][S12]
Complexidade de Código	1	2%	[S60]
Identificar Dependências de código	1	2%	[S28]
Aprendizagem de Programação	1	2%	[S36]
Nível de Stress	1	2%	[S21]
Total	63	100%	

Fonte: Elaborado pelo autor.

processo cognitivo que possui base na teoria de compreensão de programas (SOLOWAY; EHRLICH, 1984).

Estudos primários (19%, 12/63) focaram em identificar o nível de dificuldade das tarefas baseados em medidas da carga cognitiva. Identificando o nível de dificuldade das tarefas para os desenvolvedores pode ser um indicador essencial nas atividades de engenharia de software, tais como alocar recursos humanos para diferentes tarefas, ou prevenir a inserção de *bugs* (MÜLLER; FRITZ, 2016). Pesquisadores também poderiam utilizar a dificuldade de tarefas para predizer o tempo para completar uma tarefa (LEE et al., 2017).

Além disso, cerca de 10% (6/63) dos estudos primários tiveram o objetivo de medir a carga cognitiva dos desenvolvedores para identificar interrupção no trabalho. Alguns estudos (3%, 2/63) investigaram as emoções dos desenvolvedores durante atividades de desenvolvimento de software e investigou níveis de estresse dos desenvolvedores. Uma minoria dos estudos primários focou em utilizar dados da carga cognitiva para avaliar a produtividade dos desenvolvedores (5%, 3/63). A interrupção do trabalho é um fator relevante em direção a melhorar produtividade dos desenvolvedores. A interrupção de um desenvolvedor concentrado em uma tarefa prejudica sua capacidade de retomada da tarefa e sua produtividade. Futuras pesquisas poderiam focar neste propósito para entender os motivos que desenvolvedores gastam mais carga cognitiva para concluir suas atribuições. Estudos primários também focaram pouco em relação ao estado

emocional dos desenvolvedores. A falta de conhecimento em relação aos estados emocionais limita o esclarecimento de motivos de como de como o estresse e emoções negativas impactam a carga cognitiva dos desenvolvedores de software. Estas investigações podem ser cruciais para melhorar a produtividade.

Além disso, estudos primários também investigaram carga cognitiva com o propósito de rastrear dependências do código fonte (2%, 1/63), estimar complexidade de código (2%, 1/63), e aprendizagem de programação (2%, 1/63). Cerca de 6% (4/63) dos estudos primários focaram em identificar quando os desenvolvedores de software estão propensos a inserir problemas de qualidade no código fonte, tais como *bugs*. Alguns estudos primários (5%, 3/63) estimaram o nível de conhecimento de desenvolvedores através da atividade das ondas cerebrais (CRK; KLUTHE; STEFIK, 2015; CRK; KLUTHE, 2016).

Poucos estudos investigaram problemas de qualidade no código fonte, a produtividade dos desenvolvedores e mediram a complexidade de código baseado na carga cognitiva dos desenvolvedores. Estes propósitos têm o potencial de transformar a indústria de software. A indústria de software atualmente se baseia em métricas tradicionais de software, tais como a complexidade cognitiva e ciclomática, as quais calcularam a quantidade de ciclos que um trecho de código realiza. Estas métricas, junto a uma longa gama de métricas de software, foram criticadas por possuir pouca correlação com compreensão de código (SCALABRINO et al., 2019), o que pode comprometer a acurácia de indicadores de desenvolvimento de software. Desenvolvedores utilizam essas métricas para apontar problemas de qualidade de código, estimar tempo de entrega dos desenvolvedores, e para refatorar código quando a complexidade é alta demais. Desta forma, existe a possibilidade que esses tipos de atividades tenham algum tipo de imprecisão e então podendo prejudicar o processo de desenvolvimento de software enquanto este processo depender apenas de métricas baseados no código fonte. A carga cognitiva, que é um indicador individual e baseado nas percepções dos desenvolvedores de software tem potencial de melhorar a precisão de calcular essas tarefas com mais precisão. Por exemplo, uma alta carga cognitiva poderia apontar que um código em que desenvolvedores devem dividir em funcionalidades menores, e deste modo, diminuir a complexidade do código fonte.

Apenas 2% (1/63) dos estudos primários focaram em medir a carga cognitiva para apoiar o processo de aprendizagem em programação. O suporte a aprendizagem baseado nas informações da carga cognitiva dos estudantes poderia viabilizar o desenvolvimento de materiais aprimorados para a aprendizagem em programação. A falta de suporte a estudantes a aprendizagem de programação é um fator que pode contribuir a formação de profissionais.

3.2.6 QP6: quais tarefas foram usadas para medir a carga cognitiva dos desenvolvedores?

A questão de pesquisa 6 investigou as tarefas que os pesquisadores utilizaram para medir a carga cognitiva dos desenvolvedores as quais são apresentadas na Tabela 15.

O principal resultado é que a maioria dos estudos primários (83%, 52/63) analisaram carga

cognitiva, principalmente durante tarefas de programação. Pesquisas já apontaram também que a tarefa de programação é frequentemente utilizada nesta área de pesquisa (OBAIDELLAH; AL HAEK; CHENG, 2018; SHARAFI; SOH; GUÉHÉNEUC, 2015). Outro resultado importante nesta questão de pesquisa é que tarefas de programação geralmente são compostas por tarefas de compreensão ou de alteração no código, como apresentado na Tabela 16. Tarefas de compreensão consistem em atividades em que desenvolvedores realizam uma leitura do código fonte, e mentalmente processam as instruções desse código fonte para deduzir seu resultado. Nas tarefas de alterações de código, desenvolvedores modificam o código fonte através da extensão e adição de novas linhas. A Tabela 16 apresenta os estudos primários mapeados de acordo com essas categorias.

Tabela 15: Classificação de estudos primários em relação às tarefas de Engenharia de Software.

Tarefas	**#**	**%**	**Estudos Primários**
Programação	52	83%	[S01][S04][S05][S06][S08][S09][S10][S12][S15][S17] [S18][S19][S20][S21][S22][S24][S25][S26][S27][S28] [S29][S30][S31][S32][S33][S34][S35][S36][S37][S39] [S41][S42][S43][S44][S45][S46][S48][S49][S50][S51] [S52][S53][S54][S55][S56][S57][S58][S59][S60][S61] [S62][S63]
Modelagem	4	6%	[S02][S11][S13][S14]
Tarefas de Revisão	3	5%	[S16][S23][S47]
Cálculo de Equações Aritméticas	2	3%	[S03][S07]
Rotação de Estruturas de Dados	1	2%	[S40]
Memorização e Solução de Equações Aritméticas	1	2%	[S38]
Total	63	100%	

Fonte: Elaborado pelo autor.

A Tabela 15 apresenta que uma pequena parte dos estudos primários adotou tarefas de modelagem (4%, 4/63) para avaliar a carga cognitiva. Nessas tarefas, desenvolvedores analisaram Modelos de Processos de Negócios (BPMN). Pesquisadores também avaliaram a carga cognitiva dos desenvolvedores durante tarefas de revisão (5%, 3/63). Em alguns estudos ([S16][S47]), essas tarefas consistiram na revisão de alterações no código fonte antes de serem submetidas ao repositório. Em um dos estudos primários [S23] a carga cognitiva dos participantes foi medida durante a revisão de artefatos textuais. Cerca de 3% (2/63) dos estudos primários adotaram tarefas de equações aritméticas. Cerca de 2% (1/63) dos estudos consistiram em desenvolvedores realizando tarefas mentais para rotacionar estruturas de dados. Especificamente, essas tarefas consistiam em apresentar estruturas de dados em dimensão 3D para os desenvolvedores, os quais deveriam mentalmente reconhecer uma representação equivalente da estrutura em uma representação em 2D. Apenas 2% (1/63) dos estudos primários exigiram dos participantes uma combinação de tarefas de memorização seguidas por tarefas de equações matemáticas. O propósito foi analisar o quanto que interrupções poderiam prejudicar uma tarefa de baixa demanda cognitiva (memorização) comparado a uma tarefa que demanda uma carga cognitiva um pouco

maior (equação aritmética).

No geral, existem poucos estudos que utilizaram atividades de revisão, tarefa à qual é fundamental para assegurar a qualidade de código. Poucos estudos também adotaram tarefas de modelagem, as quais são essenciais para entender o comportamento dos desenvolvedores ao modelar arquiteturas de software. Ao invés disso, os poucos estudos que existem focaram em processos de negócios.

Além dessas limitações, os estudos primários não consideram investigar outras atividades que não foram registradas na Tabela 15, tais como a condução de testes unitários e atividades de implantação de software, tais como desenvolvedores implantando e modificando a configuração de sistemas durante a fase implantação. Desta forma, existe pouco conhecimento sobre carga cognitiva com tarefas relacionadas ao ciclo de desenvolvimento software em geral. O outro aspecto que não foi investigado pelos estudos foi a carga cognitiva de atividades que são fundamentais para a programação. Pesquisadores em engenharia de software pouco exploraram atividades tais como o cálculo de equações matemáticas, estrutura de dados, e memorização de artefatos. A tarefa de programação é uma tarefa complexa que envolve essas tarefas fundamentais. Esta é uma indicação de que estudos na área de engenharia de software estão negligenciando etapas relevantes para entender a carga cognitiva em relação as tarefas de programação. Desta forma, seria importante que mais estudos buscassem entender como a carga cognitiva dos desenvolvedores varia ao resolver equações matemáticas, compreensão de estruturas de dados, e em tarefas de memorização. A soma da carga cognitiva que os desenvolvedores gastam nestas tarefas poderiam contribuir de forma granular a contabilização do esforço mental nas tarefas de programação.

Em geral, os estudos primários focaram em tarefas de programação. A Tabela 16 apresenta os estudos primários mapeados de acordo com as categorias anteriormente descritas. Como observado na Tabela 16, a maioria das tarefas de programação são tarefas de compreensão de código fonte. Apenas um estudo primário considerou ambas as tarefas, i.e., compreensão e alteração do código fonte. A pesquisa de MÜLLER (2015) utilizou ambas as tarefas porque o primeiro tipo de tarefa foi utilizado para investigar a viabilidade de classificar o nível de dificuldade de tarefa durante tarefas de compreensão. O segundo tipo de tarefas foi utilizado para investigar a possibilidade de classificar dificuldade durante tarefas de alteração de código fonte.

Enquanto existe uma alta concentração de estudos que realizaram tarefas de programação, uma quantidade muito pequena de estudos adotou tarefas de alteração de código fonte, i.e., extensão e adição de funcionalidades ao código fonte. Essa quantidade de estudos primários em tarefas de alterações indica a dificuldade de implementar um ambiente para executar a avaliação deste tipo de tarefa. Isto porque, poucos estudos utilizaram algum tipo de solução ou ferramenta para integrar os dados dos sensores no ambiente de desenvolvimento. Outro fator é a dificuldade para realizar uma análise neste tipo de tarefa desde que tarefas de extensão de código possuem maior tempo de duração. Além disso, são tarefas que exigem mais mobilidade

Tabela 16: Classificação de estudos primários de acordo com os tipos de tarefas de programação.

Tarefas de Programação	**Estudos Primários**
Tarefas de Alteração	[S08][S10][S12][S18][S19][S21][S24][S27][S32][S39][S42] [S53][S55][S57]
Tarefas de Compreensão	[S01][S04][S05][S06][S09][S15][S17][S20][S22][S25][S26] [S28][S29][S31][S33][S34][S35] [S36][S37][S41][S43][S44] [S45][S46][S48][S49][S50][S51][S52] [S54][S56][S58][S59] [S60][S61][S62][S63]
Tarefas de Compreensão e Alteração	[S30]

Fonte: Elaborado pelo autor.

dos desenvolvedores, o que pode aumentar a inserção de ruídos nos dados psicofisiológicos, i.e., movimentos de mãos e braços.

3.2.7 QP7: quais foram os artefatos usados nas tarefas cognitivas?

A questão de pesquisa 7 investigou quais artefatos que os pesquisadores utilizaram para avaliar a carga cognitiva no campo de pesquisa de engenharia de software. A Tabela 17 apresenta a classificação dos estudos primários em relação aos artefatos utilizados nas tarefas. O principal resultado é o foco dos estudos primários em adotar o código fonte como principal artefato. Deste modo, 81% (51/63) dos estudos primários adotaram trechos de código fonte nas tarefas experimentais.

Tabela 17: Classificação dos estudos primários em relação aos artefatos.

Artefatos utilizados	**#**	**%**	**Estudos Primários**
Código Fonte	51	81%	[S01][S04][S05][S06][S08][S09][S10][S12][S15][S16] [S17][S18][S19][S20][S21][S22][S24][S25][S26][S27] [S28][S29][S30][S31][S32][S33][S34][S35][S36][S37] [S39][S41][S42][S43][S44][S46][S48][S49][S50][S51] [S52][S53][S54][S55][S56][S58][S59][S60][S61][S62] [S63]
Modelos de Processo	4	6%	[S02][S11][S13][S14]
Equação	2	3%	[S03][S07]
Texto	2	3%	[S23][S47]
Código Fonte & Linguagem Natural	2	3%	[S45][S57]
Estruturas de Dados	1	2%	[S40]
Equações & Polígonos	1	2%	[S38]
Total	63	100%	

Fonte: Elaborado pelo autor.

Uma minoria de estudos primários também utilizou Modelos de Processos de Negócio (4%, 4/63) e equações matemáticas (3%, 2/63). Uma minoria de estudos também utilizou artefatos

tais como textos (3%, 2/63), combinação de equações e polígonos (2%, 1/63), estruturas de dados (2%, 1/63), e combinação de código fonte com texto (2%, 1/63). A presença de artefatos tais como Modelos de Processos de Negócio e estrutura de dados indicam que estudos primários buscam explorar artefatos de diferentes áreas relacionadas ao desenvolvimento de software. Porém, outros artefatos responsáveis por compor um processo de desenvolvimento de software não estão listados na tabela, e.g., diagramas de classe, de sequência, e de estados. Os estudos primários não cobrem a totalidade de variações de artefatos envolvidos no processo de desenvolvimento de software.

Por exemplo, 4% (4/63) dos estudos primários utilizaram diferentes modelos de processo para mensurar a carga cognitiva. Um estudo primário aplicou vários tipos de notações de modelos de processo, tais como BPMN, eGantt, EPC, e Petri Net ([S02]). Outros estudos primários adotaram apenas o diagrama BPMN ([S11] [S13] [S14]).

Apenas um estudo primário utilizou estruturas de dados para avaliar a carga cognitiva. O artefato consistia em uma árvore binária. Este estudo primário não utilizou uma série de estruturas de dados utilizadas na área de engenharia de software, tais como árvores AVL. É necessário investigar como esses artefatos demandam carga cognitiva pois a programação é uma tarefa que engloba muitos tipos de estruturas de dados.

Além disso, as equações avaliadas nos estudos primários não é uma amostragem representativa de uma vasta gama de equações matemáticas e fórmulas utilizadas para desenvolver softwares. Os estudos primários focaram em avaliar cálculos simples de aritmética, e esses problemas não testam o domínio de conhecimento dos participantes. Softwares consistem em várias equações complexas de acordo com o domínio do problema. Por exemplo, diferentes fórmulas contábeis compõe um software de gerenciamento de comércio eletrônico. A falta de conhecimento dos desenvolvedores nessas fórmulas nessas equações matemáticas pode impactar na carga cognitiva. De modo geral, estudos primários não focaram em equações focadas em domínios específicos.

Apenas dois estudos primários utilizaram textos como artefatos. Projetos de engenharia de software incluem documentos de especificação tais como, a descrição do projeto com requisitos mínimos, documentos que contém especificações do software desenvolvido, e documentações de linguagens de programação.

Os artefatos utilizados foram geralmente pequenos, e contendo poucos componentes, e.g., uma árvore binária contendo até cinco nodos, uma equação matemática com até três operadores, ou código fonte com no máximo 10 linhas. Pesquisadores buscaram controlar fatores nos artefatos que poderiam tornar a tarefa mais difícil, tais como a presença e ausência de *bugs* no código fonte (CASTELHANO et al., 2019), e a presença ou ausência de identificadores disruptivos (FAKHOURY et al., 2018). Estudos primários justificam o tamanho da tarefa a viabilizar que as tarefas possam ser concluídas pelos participantes.

A Tabela 18 apresenta quais linguagens foram utilizadas por estudos primários que adotaram o código fonte. A linguagem de programação Java foi a mais utilizada para representar o

Tabela 18: Lista de linguagens de programação utilizadas nos estudos primários.

Linguagens de Programação	**Estudos Primários**
Java	[S01][S04][S08][S09][S10][S12][S16][S17][S18][S19][S22] [S24][S27][S29][S30][S34] [S35][S36][S42][S43][S46][S52] [S54][S56][S58][S59][S60][S61][S62]
Suporte a qualquer linguagem	[S15][S20][S21][S63]
C#:	[S06][S37]
C\C++	[S40][S44][S49][S50]
Linguagem Natural e Linguagem C	[S25][S45][S57]
Java e C++:	[S05]
C#, C++, e Java	[S28]
Pseudo-código e Java	[S26]
Pseudo-código	[S48]
Não especificou	[S31][S32][S33] [S51][S53][S55]

Fonte: Elaborado pelo autor.

código fonte. Em geral, pesquisadores adotaram a linguagem Java por causa da sua frequente utilização na comunidade de desenvolvedores. Também foi reportado que antes de desenvolver as tarefas, foi consultado a partir dos participantes que a linguagem de maior domínio era Java. A ampla utilização de Java nos experimentos também foi um padrão encontrado na literatura (SHARAFI; SOH; GUÉHÉNEUC, 2015; OBAIDELLAH; AL HAEK; CHENG, 2018). Poucos estudos ([S40] [S45]) analisaram o impacto de diferentes artefatos de software, i.e., entre artefatos textuais e o código fonte em relação aos indicadores de carga cognitiva. Por exemplo, FLOYD; SANTANDER; WEIMER (2017) evidenciaram que diferentes regiões cerebrais são ativadas quando os desenvolvedores interagem com código fonte em relação a pseudocódigos.

Os estudos primários em geral adotaram a linguagem Java porque os participantes eram familiarizados com esta linguagem. O objetivo dos pesquisadores era utilizar uma linguagem de programação para evitar que os participantes aplicassem carga cognitiva extra com aspectos da linguagem ao invés do domínio do problema. Outra observação é que as linguagens de programação adotadas pelos estudos primários não refletem a realidade dos projetos de software. A Tabela 18 não apresenta várias linguagens de programação que geralmente compõe um projeto de software real. Por exemplo, não foram utilizados nos estudos primários linguagens de programação tais como PHP, Javascript, MySQL, e APIs tais como Angular e NodeJS. A falta de estudos adotando esses artefatos evidenciam uma lacuna entre a academia e as práticas adotadas na indústria de software.

3.2.8 QP8: quantos participantes os estudos recrutaram para medir a carga cognitiva?

A questão de pesquisa 8 investigou a quantidade de participantes recrutados pelos estudos primários nos seus experimentos para medir a carga cognitiva na engenharia de software. Essa questão é respondida através da classificação dos estudos em cinco faixas diferentes de quan-

tidade de participantes. A Tabela 19 apresenta a classificação dos estudos primários de acordo com os intervalos em respectivo a quantidade de participantes recrutados pelos estudos primários.

Tabela 19: Classificação dos estudos primários baseados na quantidade de participantes.

Quantidade de Participantes	**#**	**%**	**Estudos Primários**
0-10	13	21%	[S19][S23][S24][S29][S35][S41][S44][S50][S51][S53] [S55][S58][S62]
11-20	20	32%	[S01][S03][S05][S06][S07][S08][S17][S21][S22][S26] [S30][S36][S37][S38][S39][S42][S43][S47][S49][S52]
21-30	15	24%	[S02][S10][S18][S20][S25][S27][S31][S34][S45][S46] [S56][S57][S59][S60][S61]
31-40	5	8%	[S04][S09][S14][S32][S33]
41-50	2	3%	[S12][S16]
70-80	3	5%	[S11][S13][S40]
Does not specify	5	8%	[S15][S28][S48][S54][S63]
Total	63	100%	

Fonte: Elaborado pelo autor.

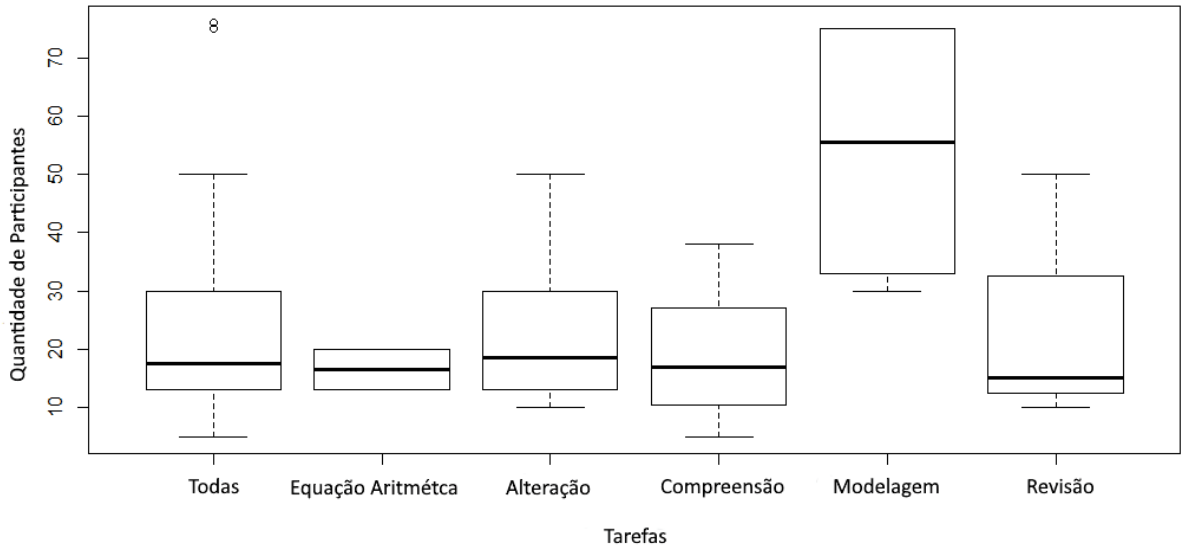
Grande parte dos estudos primários (53%, 33/63) recrutou menos de 20 participantes para medir a carga cognitiva. Alguns estudos (21%, 13/63) recrutaram menos de 10 participantes. Uma parte significante dos estudos primários (24%, 15/63) recrutaram entre 21 e 30 participantes. Esta proporção de participantes é a mesma encontrada em estudos anteriores (OBAIDELLAH; AL HAEK; CHENG, 2018; SHARAFI; SOH; GUÉHÉNEUC, 2015). Cerca de 8% (5/63) dos estudos primários recrutaram entre 31 e 40 participantes; Apenas 3% (2/63) dos estudos primários recrutaram entre 41 e 50 participantes. Cerca de 5% (3/63) dos estudos primários recrutaram entre 70 e 80 participantes. Recrutar acima de 50 participantes é uma faixa elevada para esta área de pesquisa.

Especificamente, maior parte dos estudos recrutou em um intervalo entre 11 e 20 participantes. O recrutamento entre 11 e 20 participantes aparentemente forma uma amostragem pequena e limitada para esse tipo de pesquisa. Contudo, uma grande quantidade de participantes não é obrigatória para gerar uma grande amostragem nos dados. Equipamentos psicofisiológicos capturam uma grande quantidade de dados por segundo, tornando o conjunto de dados para análise de alta dimensão. Porém, a concentração de estudos nesse intervalo indica uma dificuldade dos pesquisadores em recrutar participantes para os experimentos.

Figura 7 apresenta a distribuição da quantidade de participantes por tipo de tarefa (QP6). A linha em realce representa a mediana. É possível observar que a distribuição da quantidade de participantes recrutados em pesquisas que realizaram tarefas de compreensão e alteração de código é próxima a distribuição geral (Todas). A mediana da quantidade de participantes geral (Todas) é cerca de 18 participantes, e o terceiro quartil é 30. Esta distribuição significa que em geral, a quantidade de participantes que os pesquisadores recrutaram reside no intervalo

entre 20 e 30 participantes. A distribuição dos participantes nas tarefas de compreensão e alteração de código possuem um padrão de distribuição muito parecido com a distribuição de todos os participantes. Estas tarefas são praticamente as que representam o desenvolvimento de software dentro das pesquisas de carga cognitiva na engenharia de software. O *boxplot* das tarefas de compreensão mostra que a quantidade de participantes é, em geral, um pouco menor que a quantidade de participantes recrutados nas tarefas de compreensão. Como o *boxplot* está enviesado para cima, isto indica que a distribuição não é normal. O espaço entre o terceiro quartil e o valor máximo evidencia que poucos estudos recrutaram uma quantidade entre 30 e 50 participantes. Desta forma, estudos que recrutaram mais que 50 participantes são *outliers* nesta área de pesquisa.

Figura 7: A distribuição dos participantes em relação às tarefas.



Fonte: Elaborado pelo autor.

O recrutamento de participantes é uma tarefa desafiadora. Primeiro, o tempo para recrutar participantes é limitado porque os prazos para produzir resultados de pesquisa são geralmente curtos. Além disso, a metodologia do estudo pode restringir a seleção de participantes. Por exemplo, uma quantidade de participantes seria menor se o objetivo da pesquisa fosse direcionado a entender a carga cognitiva de desenvolvedores com alguma habilidade específica. Desta forma, participantes que não possuem essa característica específica não seriam recrutados. Outro fator, é a falta de engajamento de participantes por causa da dificuldade das tarefas de programação, e da dificuldade de implantar sensores nos participantes. Os procedimentos de implantação de sensores são demorados e desta forma participantes podem ter receio de participar (RADEVSKI; HATA; MATSUMOTO, 2015).

3.2.9 QP9: quais métodos de pesquisa foram utilizados para investigar a carga cognitiva nas tarefas de desenvolvimento de software?

A questão de pesquisa 9 classifica os estudos primários de acordo com o método de pesquisa utilizado. Estes estudos primários foram classificados baseado nas categorias que WIERINGA et al. (2005) propôs. WIERINGA et al. (2005) sugeriu categorizar estudos de engenharia de software em estudos de avaliação, propostas de solução, artigos filosóficos, replicação, e pesquisa que reportam experiencia pessoal. A Tabela 20 apresenta estudos primários em relação a essas categorias.

Grande parte dos estudos primários (81%, 51/63) consistiram em estudos de validação. Duas classes de estudos de validação foram identificadas, i.e., estudos experimentais no laboratório e estudos experimentais em campo. A Tabela 21 lista estudos primários que pertencem a essas categorias. Esta classificação esclarece quais estudos primários reportam resultados obtidos em campo, i.e., em prática, e estudos que reportam resultados a partir de experimentos no laboratório. Com base nessa classificação foi encontrado que pesquisadores pouco utilizaram as medidas da carga cognitiva em cenários reais de trabalho. Alguns estudos consistiam em experimentos controlados, e outros consistiam em metodologias experimentais menos rígidas conduzidas em laboratórios. Grande parte dos estudos primários consistem em experimentos controlados e isso é devido ao interesse de investigar a carga cognitiva de maneira controlada, como por exemplo, ao investigar a relação entre carga cognitiva e artefatos de software.

Tabela 20: Classificação de estudos primários baseados no método de pesquisa.

Método de Pesquisa	**#**	**%**	**Estudos Primários**
Estudos de Validação	51	83%	[S01][S03][S04][S05][S06][S07][S08][S09][S11] [S13][S14][S16][S17][S18][S19][S21][S24][S26] [S27][S29][S30][S31][S32][S33][S34][S35][S36] [S37][S38][S39][S40][S41][S42][S43][S44][S45] [S46][S48][S49][S50][S51][S52][S53][S55][S56] [S57][S58][S59][S60][S61][S62]
Proposta de Solução	5	6%	[S15][S20][S28][S54][S63]
Estudo de Replicação	3	2%	[S22][S25][S47]
Experiência Pessoal	2	3%	[S02][S23]
Artigo Filosófico	1	2%	[S12]
Proposta de Replicação	1	2%	[S10]
Total	63	100%	

Fonte: Elaborado pelo autor.

Cerca de 6% (5/63) dos estudos primários são propostas de solução para a carga cognitiva na engenharia de software. Por exemplo, PEITEK et al. (2019) propuseram uma ferramenta que integra dados dos sensores de fMRI e *Eye-Tracking* para prestar suporte a estudos de en-

genharia de software para medir a carga cognitiva. Apenas 2% (1/63) propuseram replicar um estudo anterior, e 2% (1/63) dos estudos primários validaram teorias sobre carga cognitiva na engenharia de software. Apenas 3% (2/63) dos estudos primários eram artigos de experiência os quais reportavam experiências dos autores em relação a utilização da carga cognitiva. Apenas 3% (2/63) dos estudos primários consistiam em artigos filosóficos os quais especificam novas visões e implicações.

A Tabela 20 apresenta uma tendência dos estudos primários em adotar métodos de validação para investigar a carga cognitiva. A concentração de estudos primários nessa categoria indica que pesquisadores estão investigando a viabilidade da utilização dessas medidas. Por outro lado, a Tabela 21 apresenta apenas quatro estudos de validação realizaram avaliações em campo, i.e., avaliaram a viabilidade de medir a carga cognitiva em ambientes com rotina real de trabalho.

Baseado na quantidade de estudos que eram propostas de soluções, existe uma tendência de que a integração entre indústria e academia em relação aos estudos de carga cognitiva permaneça limitado. Pesquisadores desenvolveram ferramentas focando em análises científicas, porém não foi fornecido uma integração de soluções que podem ser utilizadas no ambiente real de trabalho para que profissionais da indústria possam replicar. Uma quantidade limitada de estudos replicou estudos anteriores. A falta de estudos de replicação é um problema conhecido na comunidade acadêmica de engenharia de software. Pesquisadores tem dificuldade em acessar conjunto de dados (*datasets*) e pacotes de replicação utilizados em pesquisas anteriores. Dados psicofisiológicos é um dado sensível e geralmente confidencial. A falta de artigos de experiencia evidencia uma limitação de estudos em engenharia de software em reportar erros e casos de sucesso vivenciados pelos autores. A falta destes estudos implica que novas pesquisas estão sujeitas a cometer os mesmos problemas que outros pesquisadores já sabem como resolver.

Por fim, pesquisadores na área de engenharia de software também produziram uma quantidade limitada artigos filosóficos. A falta de artigos filosóficos indica que pesquisadores não focaram em estruturar o conhecimento em relação a carga cognitiva na engenharia de software. Por exemplo, um *framework* tornaria viável para definir quais medidas e quais processos futuros pesquisadores poderiam utilizar para organizar investigações sobre carga cognitiva na engenharia de software.

Tabela 21: Lista de estudos primários classificados como pesquisa de avaliação.

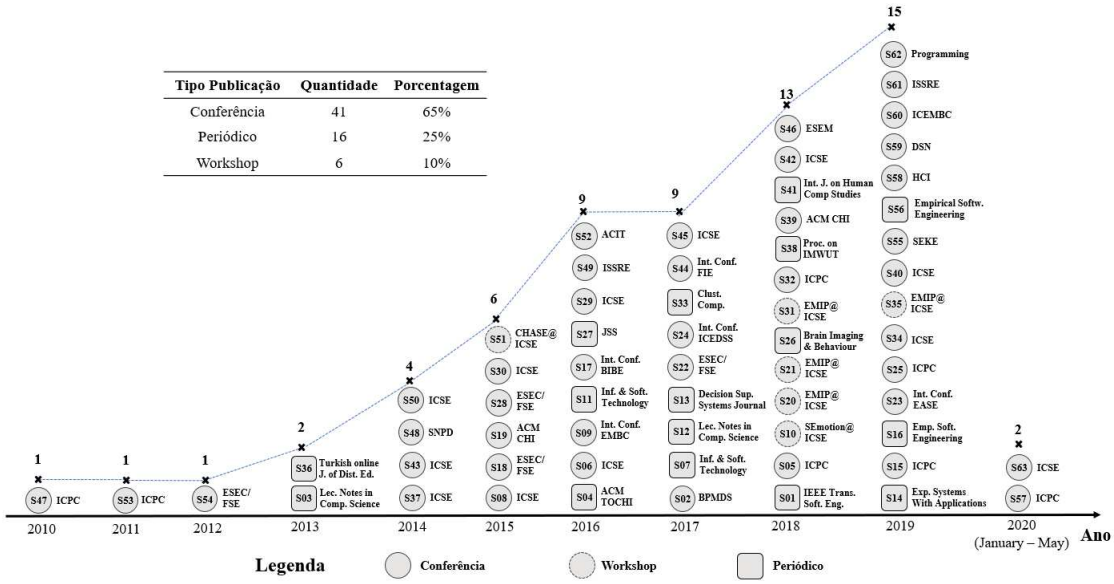
Estudos de Avaliação	**Estudos Primários**
Experimentos Controlados	[S01][S03][S04][S05][S09][S11][S13][S14][S16][S17][S18][S22] [S25][S26][S27] [S31][S40][S41][S42][S43][S44][S45][S46][S47] [S49][S56][S57][S62]
Estudos experimentais em laboratório	[S06][S07][S08][S21][S29][S34][S35][S36][S37][S38][S48][S50] [S51][S52][S53] [S55][S58][S59][S60][S61]
Estudos experimentais em campo	[S19][S24][S39][S32]

Fonte: Elaborado pelo autor.

3.2.10 QP10: onde os estudos foram publicados?

A Figura 8 apresenta uma cronologia em um intervalo de 10 anos dos estudos primários. Esta questão de pesquisa classifica os estudos como estudos publicados em conferência, periódicos, ou *workshops*.

Figura 8: Histórico de publicações dos estudos primários selecionados.



Fonte: Elaborado pelo autor.

Quantidade de publicações. Cada estudo primário publicado soma um ponto ao total a quantidade de publicações por ano. A Figura 8 contém uma linha pontilhada que acompanha que acompanha a quantidade de estudos primários publicados por ano. A maior quantidade de publicações está concentrada entre 2018 e 2019. Desta forma, é possível notar a distribuição e onde os autores disseminaram suas pesquisas. Apesar da estratégia de busca considerar estudos publicados até 2020, a pesquisa sobre carga cognitiva na área de engenharia de software significativamente aumentou após o lançamento de tecnologias tais como o NeuroSky (KATONA et al., 2014), e o Emotiv (Emotiv Systems, 2021). Estudos publicados após 2014 mencionaram a utilização dessas tecnologias frequentemente. Desta forma, a pesquisa sobre a medida da carga cognitiva tem aumentado consecutivamente.

Tendências. A quantidade de estudos publicados teve um aumento contínuo desde 2014, exceto em 2017, ano o qual manteve a quantidade de estudos de 2016. O período mais produtivo aconteceu entre 2017 até 2019, onde concentra-se 59% (37/63) das publicações dos estudos primários em conferências, periódicos, e *workshops*. Durante este período, cinco estudos primários foram publicados em *workshops*. A publicação em workshops indica uma tendência dessa área de pesquisa em discutir futuros desafios e resultados emergentes com maior frequência. Os estudos primários também foram publicados em periódicos de alto impacto relacionados a

engenharia de software [1]. Foram encontrados onze estudos (S01, S04, S07, S11, S14, S16, S26, S27, S41, S56) publicados em periódicos de alto impacto (IEEE TSE, IST, *Expert Systems with Applications*, JSS, IJHCS, e *Empirical Software and Engineering*). Esta área de pesquisa possui uma forte tendência em ser publicada em conferências relacionadas a engenharia de software, incluindo ICSE (S06, S08, S29, S30, S34, S37, S40, S42, S43, S45, S50, S63) e *International Conference on Program Comprehension* (S05, S15, S25, S32, S47, S53, S57). Além disso, 2019 foi o ano que produziu a maior quantidade de estudos primários. Estes dados reforçam que investigar a carga cognitiva na engenharia de software é uma pesquisa emergente com potencial de crescimento. Então, desta forma ainda existem desafios significativos e problemas a serem resolvidos.

Locais de Publicação. A Figura 8 também mostra que os autores tendem a publicar estudos primários em conferências mais do que em periódicos. Esta preferência pode significar que medir a carga cognitiva é um tópico de pesquisa emergente na engenharia de software. As publicações em periódicos tendem a aumentar assim que os autores estruturarem a carga cognitiva na área de engenharia de software e desenvolver novas taxonomias, e *frameworks* para avaliar a carga cognitiva na engenharia de software.

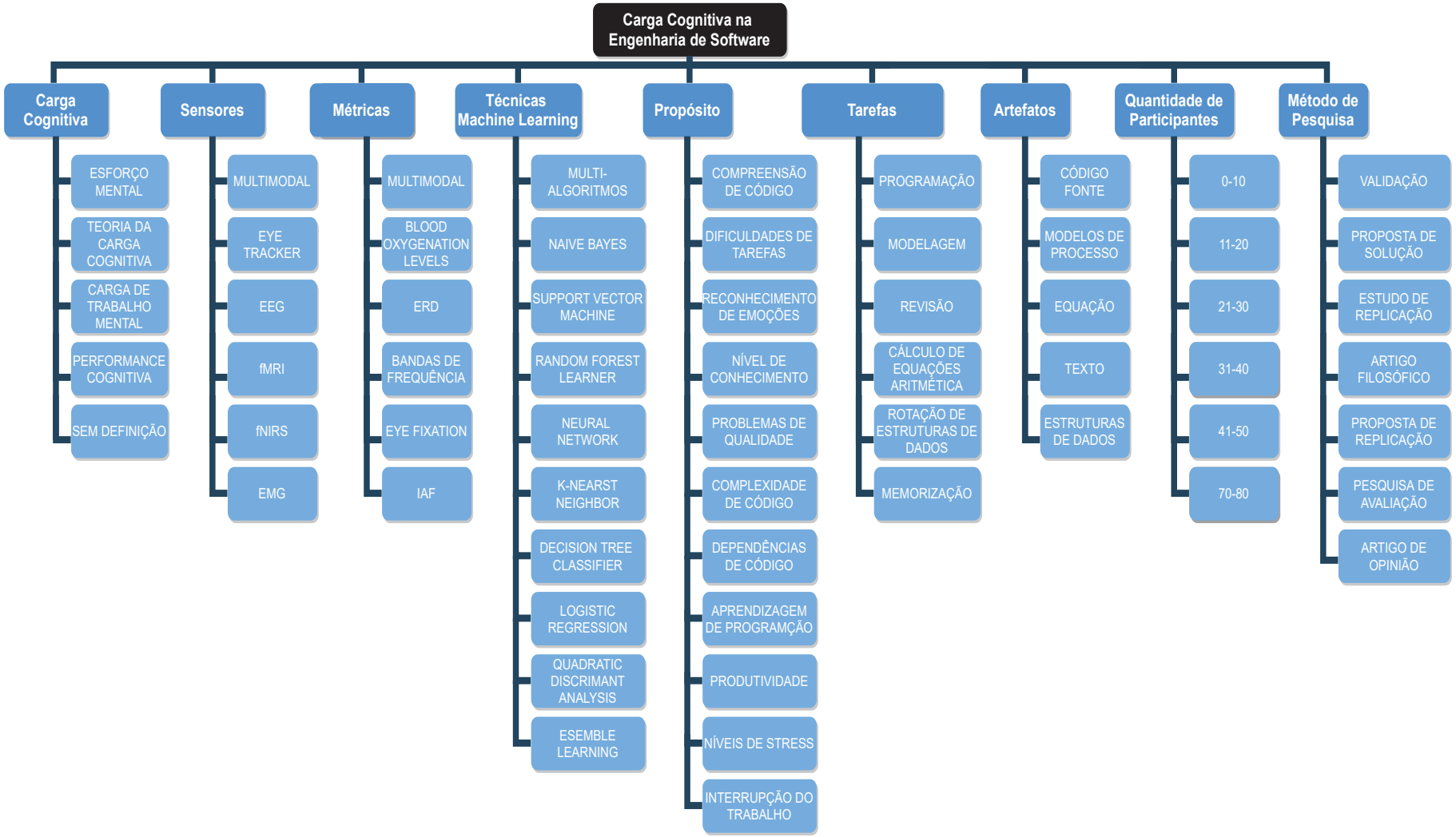
3.2.11 Esquema de classificação

A Figura 9 apresenta o esquema de classificação representado as respostas utilizadas para cada questão de pesquisa do estudo de mapeamento sistemático. Este esquema de classificação representa o conhecimento obtido através dos estudos primários analisados neste estudo de mapeamento sistemático. Este esquema de classificação agrupa as respostas as respectivas questões de pesquisa. O quadrado de cor preta destaca o domínio da pesquisa, que é carga cognitiva na engenharia de software. Os quadrados destacados com cores azuis escuro representam as principais questões de pesquisa, a partir da QP1 a QP9. O esquema de classificação organiza essas questões de pesquisa lado a lado em direção horizontal — desta forma, em direção horizontal se alterna entre as respectivas questões de pesquisa. Na direção vertical desse esquema estão descritas as repostas das questões de pesquisa em forma de subcategorias. A cor azul clara destaca essas subcategorias. Especificamente, a relação das questões de pesquisa e as categorias são as seguintes:

- **Carga Cognitiva (QP1):** as teorias utilizadas pelos estudos primários para definir o termo carga cognitiva;
- **Sensores (QP2):** os dispositivos os quais os estudos primários utilizaram para coletar dados relacionados a carga cognitiva;

[1] Periódicos com h5-médio maior que 40 de acordo com a indexação *Google Schoolar* (`https://scholar.google.com/`)

Figura 9: Esquema de classificação da carga cognitiva na Engenharia de Software.



Fonte: Elaborado pelo autor.

- **Métricas (QP3):** as medidas extraídas dos dados dos sensores utilizadas para investigar a carga cognitiva na engenharia de software;
- **Técnicas de *Machine Learning* (RQ4):** as técnicas de *machine learning* utilizadas em conjunto com dados da carga cognitiva;
- **Propósitos (QP5):** os propósitos que estudos em engenharia de software mediram a carga cognitiva dos desenvolvedores;
- **Tarefas (QP6):** as tarefas que estudos primários escolheram para avaliar a carga cognitiva dos desenvolvedores;
- **Artefatos (QP7):** os artefatos que estudos primários selecionaram para incorporar as tarefas e avaliar a carga cognitiva;
- **Participantes (QP8):** a quantidade de participantes nas quais os estudos primários recrutaram para realizar as tarefas e experimentos; e
- **Métodos de pesquisa (QP9):** os métodos de pesquisa utilizados pelos estudos primários para investigar a carga cognitiva.

O esquema de classificação na Figura 9 foi utilizado para rastrear se existem estudos que representem um padrão entre as diferentes questões de pesquisas. Foram encontrados seis estudos primários ([S05] [S31] [S46] [S56] [S59] [S61]) que consistiam em estudos de validação, utilizam sensores multimodais e métricas multimodais. Estes estudos também definiram a carga cognitiva como sinônimo de esforço mental. Estes estudos também recrutaram entre 15 e 30 participantes para tarefas de programação. Esses estudos também utilizaram a carga cognitiva com o propósito de investigar a compreensão de código ([S05] [S46] [S56]), dificuldades de tarefas ([S31]), problemas de qualidade de código ([S61]), e classificação da carga cognitiva ([S59]). Outro fator em comum entre esses estudos primários é a não utilização de técnicas de *machine learning* com os dados psicofisiológicos. Estes estudos focaram em correlacionar áreas de ativação do cérebro com atividades de engenharia de software. É evidente que alguns estudos primários utilizaram recursos de imagens cerebrais tais como fNIRS e fMRI com o *Eye-Tracker* para identificar a variação de ativação em áreas cerebrais durante a interação em tarefas relacionadas a engenharia de software ([S05] [S31] [S46] [S56]). Outro agrupamento de estudos primários encontrados é aquele que analisa áreas cerebrais em atividades de programação. Estes estudos adotaram exclusivamente o fMRI ([S01] [S22] [S43] [S49]) para investigar compreensão de código ([S01] [S22] [S43]) e problemas de qualidade de código ([S49]).

Pesquisadores e praticantes na indústria podem se beneficiar a partir de várias formas deste esquema de classificação. Estes benefícios estão listados abaixo:

Principais benefícios para pesquisadores: baseado neste esquema de classificação, pesquisadores poderiam classificar estudos futuros sobre medidas da carga cognitiva na engenharia

de software, e replicar a pesquisa conduzida neste capítulo com o objetivo de destacar novas soluções, e futuros desafios. Pesquisadores poderiam utilizar este esquema como um ponto de partida para adicionar novas terminologias relacionadas a carga cognitiva na engenharia de software. Por fim, pesquisadores podem utilizar termos deste esquema para conduzir futuros estudos empíricos.

Principais benefícios para a indústria de software: para a indústria, existe um grande interesse em utilizar a carga cognitiva para melhorar o bem estar dos desenvolvedores, produtividade, e a qualidade dos artefatos de software. Desta forma, desenvolvedores e analistas se beneficiariam a partir desse esquema evitando esforço para listar tecnologias relacionados a medidas de carga cognitiva aplicadas na engenharia de software. A indústria de software poderia construir soluções para reduzir a alta carga cognitiva. Reduzir a carga cognitiva dos desenvolvedores poderia contribuir para melhorar a produtividade dos desenvolvedores.

3.3 Discussão dos resultados

Esta seção apresenta uma discussão e outros desafios que foram identificados em estudos primários selecionados.

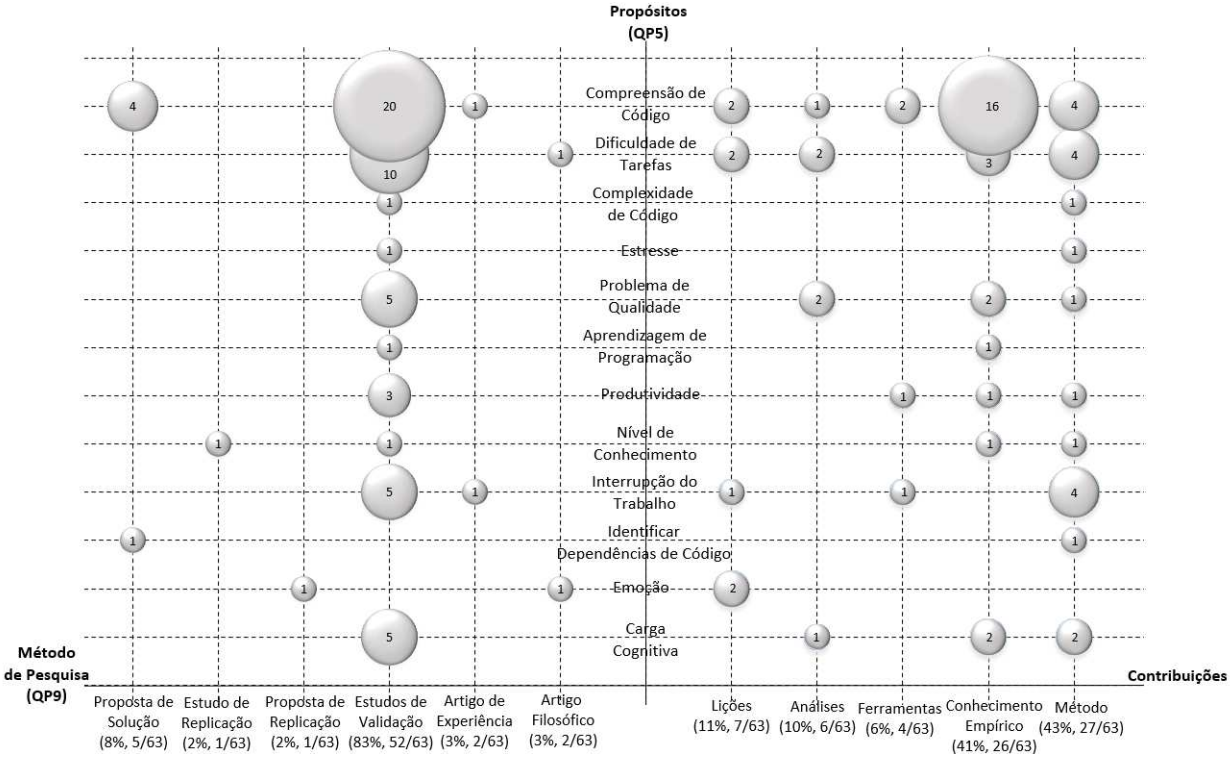
3.3.1 Gráfico de bolhas

A Figura 10 apresenta um gráfico de bolhas, que organiza os estudos primários através de três dimensões. Cada bolha possui três dimensões (d_1, d_2, d_3). Onde d_1 representa o método de pesquisa usado, quanto o tipo de contribuição realizada para medir a carga cognitiva na engenharia de software. A dimensão d_2 representa os propósitos pelos quais os estudos primários mediram a carga cognitiva na engenharia de software. Além disso, d_3 consiste na quantidade de estudos, o qual impacta no tamanho de cada bolha. Em geral, este gráfico de bolhas apresenta uma visão geral das relações entre os métodos de pesquisa usados (QP9), propósitos (QP5) e as contribuições.

O gráfico de bolhas apresentado na Figura 10 apresenta que a distribuição dos estudos primários se concentra nos quadrantes superiores tanto em relação aos exidos dos métodos de pesquisa (QP9) e das contribuições. Especificamente, os estudos que medem a carga cognitiva na engenharia de software são estudos de validação que contribuíram com conhecimento empírico com o objetivo de investigar principalmente a compreensão de código e dificuldades de tarefas.

De acordo com a Figura 10, estudos primários não exploraram e avaliaram alguns propósitos que são relevantes. Por exemplo, estudos primários que mediram a carga cognitiva para investigar complexidade de código apenas contribuiu com um método o qual os autores avaliaram através de um método de validação. Estimar a complexidade de código através da carga cognitiva dos desenvolvedores é uma meta ambiciosa. De acordo com as lacunas apresentadas na

Figura 10: Gráfico de bolhas que relaciona três variáveis. O *eixo-x* consiste no método de pesquisa utilizado (A) e nas contribuições (B). O *eixo-y* representa o propósito dos estudos primários. O tamanho das bolhas consiste na quantidade de estudos primários classificados de acordo com os critérios do *eixo-x* e *eixo-y*.



Fonte: Elaborado pelo autor.

Figura 10, é evidenciado que pesquisadores precisam produzir conhecimento práticos e teóricos para viabilizar estimar a complexidade de código através da carga cognitiva dos desenvolvedores. Esta observação inclui outros propósitos que podem potencialmente impactar a indústria de software, problemas de qualidade de código, aprendizado em programação, interrupção do trabalho, e identificação de dependências de código. As dimensões teóricas dizem respeito a teorias que precisam surgir para dar suporte a esses propósitos. Por exemplo, novas propostas por métodos, criação de *frameworks* e taxonomias a partir de artigos filosóficos, e a obtenção de conhecimento empírico após validar métodos propostos. A dimensão prática diz respeito a integração de ferramentas na indústria e o conhecimento que pesquisadores podem derivar disso. Por exemplo, pesquisadores poderiam extrair novos conhecimentos empíricos a partir de estudos de avaliação na indústria e produzir lições apreendidas a partir dessa experiência.

A Figura 10 apresenta uma forte tendência para que estudos primários conduzam estudos de validação da carga cognitiva em ambiente de laboratório. Outro fator que reforça esse viés é a falta de estudos que avaliem as ferramentas produzidas na indústria de software. Este viés está presente até mesmo em estudos primários que possuem o objetivo de resolver compreensão de programas e dificuldade de tarefas. A academia produziu uma quantidade reduzida de ferramentas e estudos de avaliação em cenários realísticos. Isto é uma tarefa viável pois existe uma

quantidade de estudos que contribuíram com conhecimento empírico e estudos de validação na academia.

3.3.2 Contribuições

Os estudos primários selecionados focaram em contribuir principalmente em cinco aspectos da carga cognitiva na engenharia de software. Essas contribuições consistem em lições apreendidas, análises, ferramentas, conhecimento empírico, e métodos para medir a carga cognitiva. As contribuições dos estudos primários estão descritas a seguir:

Lições apreendidas: alguns estudos primários reportaram lições apreendidas baseadas nas experiências dos autores em relação a experimentos que realizaram com carga cognitiva ([S02] [S06] [S10] [S12] [S20] [S23] [S30]). ZIMOCH et al. (2017) [S02] listou lições apreendidas sobre experimentos anteriores que utilizaram *Eye-Tracking* durante a compreensão de modelos de processos. Os autores listaram que o nível de conhecimento dos participantes é evidente enquanto o nível de dificuldade das tarefas aumenta. Ambos os estudos [S02] [S23] reportaram que o contexto das tarefas pode influenciar a percepção dos participantes e, então, negativamente afetar seus desempenhos. Por exemplo, uma tarefa de programação formulada na linguagem C para desenvolvedores Java implicaria em coletar resultados em desenvolvedores inexperientes em linguagem C. Outro relato importante foi que pesquisadores afirmaram que a análise a partir de dados que foram coletados de sensores multimodais possibilita análises detalhadas sobre a carga cognitiva [S06] [S10] [S12]. Contudo, Molléri et al. [S23] sugere um cuidado com os dispositivos psicofisiológicos por causa das leituras incompletas que o autor obteve a partir de um equipamento de EEG.

Análises: alguns estudos primários conduziram uma análise comparativa em relação a efetividade das medidas da carga cognitiva ([S03] [S07] [S29] [S31] [S35] [S62]). Nourbakhsh et al. [S03] analisou o impacto que a quantidade de piscares de olhos e resposta galvânica da pele (GSR) na classificação da carga cognitiva. Os autores concluíram que ao combinar GSR e quantidade de piscares de olhos impacta em maior precisão dos classificadores do que os utilizar individualmente para classificar níveis da carga cognitiva. Borys et al. [S07] concluíram que o tamanho da dilatação da pupila e EEG não foram relevantes para mensurar a carga cognitiva. A correlação do tamanho da dilatação da pupila e o nível da carga cognitiva dos desenvolvedores foi baixa. Essa baixa correlação aconteceu devido ao nível do brilho da tela do computador e do ambiente. Isto diminuiu as pupilas dos desenvolvedores implicando em menores variações da pupila durante as tarefas. Contudo, Peitek et al. [S31] evidenciaram que a dilatação da pupila variou até mesmo com o brilho da tela. DURAISINGAM; PALANIAPPAN; ANDREWS (2017) também discordou de Borys et al. [S07], apresentando a efetividade do EEG em medir a carga cognitiva. Os resultados de Ishida et al. [S35] e Uwano et al. [S62] também divergiram de Borys. Ishida et al. [S35] e Uwano et al. [S62] que evidenciaram a efetividade do EEG para medir a carga cognitiva. Esta efetividade pode ter sido consequência da aplicação de fil-

tros no sinal de EEG e de remoções de componentes tais como, componentes de movimentos musculares que não eram relacionados a carga cognitiva.

Ferramentas: consistem em estudos primários que contribuíram com ferramentas de software para dar suporte a medição da carga cognitiva para pesquisas e aplicações práticas ([S15] [S38] [S51] [S63]). Alguns estudos primários focaram em propor ferramentas para viabilizar experimentos envolvendo carga cognitiva dos desenvolvedores na área de engenharia de software, tais como a CodersMUSE [S15] e VITALISE [S63]. Os estudos primários também forneceram soluções que podem ser aplicadas na prática pela indústria. Flowlight [S32] e Collins [S38] tem o objetivo de gerenciar interrupção do trabalho dos desenvolvedores baseado em indicadores da carga cognitiva. Estes sistemas apontam quando um desenvolvedor pode ser interrompido ou não. Emendo [S51] é um modelo que tem o objetivo de monitorar carga cognitiva dos desenvolvedores em tempo real para gerenciar e monitorar produtividade dos desenvolvedores.

Conhecimento Empírico: alguns estudos primários contribuíram com conhecimentos empíricos. Estes estudos focaram em relacionar efetivamente medidas da carga cognitiva ao contexto de engenharia de software ([S01] [S04] [S05] [S09] [S11] [S13] [S16] [S17] [S22][S26] [S36] [S40] [S41] [S43] [S44] [S45] [S46] [S47] [S48] [S49] [S50] [S52] [S53] [S56] [S57] [S59]). Por exemplo, alguns estudos primários forneceram forte evidência de áreas cerebrais específicas que são ativadas enquanto desenvolvedores compreendem código fonte. PEITEK et al. (2018a) [S01], e SIEGMUND et al. (2017) [S22] encontraram áreas cerebrais relacionadas ao processamento de linguagem (BA 21, BA 44, BA 47), atenção (BA 6), e memória de trabalho (BA 6, BA 40) ativado durante tarefas de compreensão de código. CASTELHANO et al. (2019) investigaram a ativação de áreas cerebrais relacionados a detecção de *bugs* no código fonte. As mesmas áreas cerebrais também foram encontradas em relação ao estudo de PEITEK et al. (2018a) [S01] e SIEGMUND et al. (2017) [S22]. Além disso, CASTELHANO et al. (2019) também encontraram áreas cerebrais relacionadas ao processamento matemático (BA 9). Essas áreas não foram encontras no estudo de PEITEK et al. (2018a)[S01], e SIEGMUND et al. (2017)[S22] porque a estrutura das tarefas experimentais não exigia que os desenvolvedores processassem equações matemáticas. Em geral, SIEGMUND et al. (2017)[S22] isso também forneceu um *framework* para investigar compreensão de programas com sensores fMRI. Além disso, Krueger et al. [S57] e Floyd, Santander e Weimer [S45] encontraram áreas cerebrais as quais são ativas durante a leitura de códigos fonte. Ambos os pesquisadores encontraram diferentes padrões de ativação cerebral entre interpretar um código fonte e realizar leituras de textos simples.

Método: nesta categoria de contribuição estão estudos primários que contribuíram principalmente com métodos ou abordagens para medir a carga cognitiva na engenharia de software ([S08] [S14] [S18] [S19] [S21] [S24] [S25] [S27] [S28] [S32] [S33] [S34] [S37] [S39] [S42] [S54] [S55] [S58] [S60] [S61]). Estes estudos principalmente sistematizaram maneiras de relacionar artefatos de software e a carga cognitiva dos desenvolvedores. Em geral, estes estudos

contribuíram com vários métodos para medir a carga cognitiva na engenharia de software. Consistem em estudos primários que focaram em treinar abordagens de *machine learning* com dados psicofisiológicos para finalidades em engenharia de software. Müller & T. Fritz 2015 [S08] foram capazes de classificar emoções de desenvolvedores (positivas e negativas) com 71,36% de precisão. Existem contribuições relacionadas a métodos para identificar problemas de qualidade no código fonte. Müller & Fritz 2016 [S24] [S61] treinaram uma técnica *Random Forest* utilizando dados de temperatura da pele, variabilidade de taxa do batimento cardíaco, e taxa de respiração para classificar tipos de problemas de qualidade. Couceiro et al. [S61] combinou dados do *Eye-Tracker* com taxa de variabilidade de batimento cardíaco para desenvolver um método para indicar informações de carga cognitiva relacionado a partes específicas do código fonte.

3.3.3 Oportunidades e desafios futuros de pesquisa

Esta seção apresenta algumas oportunidades e desafios de pesquisa derivados dos estudos primários e das contribuições analisadas encontradas na seção 3.3.2. Os desafios e oportunidades de pesquisas identificados são os seguintes:

(1) Um método para medir a carga cognitiva em tarefas de compreensão de código. Estudos na literatura de engenharia de software aplicaram vários métodos para medir a carga cognitiva de desenvolvedores de software. Pesquisadores geralmente mediram a carga cognitiva dos desenvolvedores através de uma gama de métricas utilizando dispositivos multimodais. Esta gama de métricas utilizadas demonstra que medir a carga cognitiva dos desenvolvedores é válido através de várias fontes e sinais biológicos. Entre essas medidas, foi utilizado o *Event-Related Desynchronization* (ERD). CRK; KLUTHE (2016) calcularam o ERD baseado nas ondas cerebrais dos desenvolvedores. CRK; KLUTHE (2016) obteve o ERD calculando a média de dessincronização das ondas cerebrais aplicada durante tarefas cognitivas em relação a períodos em que as ondas estão sincronizadas, isto é, durante períodos de descanso ou inatividade. Desta forma, existem métodos para medir carga cognitiva na engenharia de software. A existência de métricas de carga cognitiva viabiliza que pesquisadores construam e proponham um método específico para mensurar carga cognitiva na engenharia de software.

Desta forma, pesquisadores poderiam focar em propor métodos ou abordagens para medir a carga cognitiva dos desenvolvedores para os propósitos listados na Figura 10. Além disso, também são necessárias novas abordagens nos propósitos tais como o de compreensão de código. Isto porque, apesar da Figura 10 sugerir que exista uma concentração de métodos e abordagens direcionadas para medir a carga cognitiva nas tarefas de compreensão de código, abordagens como as utilizadas em CRK; KLUTHE (2016), LEE et al. (2017) , KOSTI et al. (2018), DURAISINGAM; PALANIAPPAN; ANDREWS (2017), e SIEGMUND et al. (2017) adaptaram indicadores da neurociência e não são específicas para mensurar carga cognitiva em tarefas de compreensão de código. Em geral, todas essas abordagens utilizaram diferentes dispositivos,

tais como, o EEG e o fMRI, e a partir desses dispositivos foram utilizados variados indicadores, tais como o ERD, ASR, potências de frequências de banda, tais como alfa e beta, e o PLV para mensurar a carga cognitiva. Portanto, existe o risco dessas pesquisas estarem utilizando indicadores que não são apropriados para esses propósitos. Dessa forma, há uma oportunidade de analisar as características dos indicadores já utilizados e construir uma abordagem para penalizar aspectos específicos e relacionados a compreensão de código, já que, as abordagens utilizadas foram aplicadas sem um critério de escolha. Esta oportunidade de pesquisa será explorada no Capítulo 4.

(2) Analisar a correlação de indicadores psicofisiológicos com tarefas de compreensão de código. Compreensão de código é uma tarefa central na engenharia de software o qual ocupa grande parte do tempo dos desenvolvedores, pois uma boa compreensão viabiliza atividades de manutenção, revisão, ou criação de novas funcionalidades para o software. Deste modo, é importante analisar a compreensão de código, para viabilizar análises mais confiáveis em outras domínios. Analisar a correlação dos indicadores psicofisiológicos é necessário para determinar quais indicadores tem uma melhor eficiência para analisar a compreensão de código. Uma gama de indicadores foi utilizada sem a avaliação de correlação com a compreensão de código. Desta forma, foram em geral aplicadas e adotadas de maneira arbitrária.

Desta forma, pesquisadores não exploraram a correlação de métricas relacionadas a ondas cerebrais, batimentos cardíacos, taxa de respiração, e temperatura da pele em relação a compreensão de código. Na verdade, pesquisadores tem o desafio de preencher essa lacuna dessas análises de correlação em relação a todos os propósitos que não constam estudos na Figura 10. A Figura 10 evidencia uma falta de análises para praticamente a maioria dos propósitos. Como a tarefa de compreensão de código é crítica para a maioria dos demais propósitos na Figura 10, este trabalho analisou a correlação de abordagens de EEG tradicionais em relação a tarefas de compreensão de código, em particular no Capítulo 5. Especificamente, foi analisada a correlação da abordagem desenvolvida CogEff, e de abordagens de EEG para mensurar a carga cognitiva. O EEG foi utilizado porque é uma tecnologia com maior potencial de ser adotada em ambientes de engenharia de software, e ao mesmo tempo, captura dados relacionados a atividade neural dos desenvolvedores de software.

(3) Conhecimento empírico sobre a efetividade de técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código. Conduzir estudos empíricos na engenharia de software é uma tarefa desafiadora (SIEGMUND, 2016). Grande parte das contribuições na área de engenharia de software são estudos empíricos. Em relação a Figura 10, faltam conhecimentos empíricos sobre processos cognitivos de desenvolvedores de software em níveis de dificuldade, complexidade de código, níveis de stress, interrupção do trabalho, identificação de dependência de código, e os efeitos de emoções em ambientes de desenvolvimento de software. Desta forma, futuros estudos empíricos poderiam focar nestes propósitos na área de engenharia de software.

A maioria dos estudos empíricos se concentraram em compreensão de código, tratando-se

majoritariamente de experimentos controlados que verificaram a ativação de áreas cerebrais durante a tarefa compreensão de código. Isto evidencia a maturidade e profundidade com que estudos empíricos relacionaram a compreensão de código com dinâmicas cerebrais. Por outro lado, em relação a estudos empíricos sobre compreensão de código, ainda existe a oportunidade de analisar a efetividade que técnicas de *machine learning* possuem ao serem treinadas com abordagens de EEG para classificar compreensão de código. Isto porque, estudos que utilizaram técnicas de *machine learning* com dados psicofisiológicos se concentraram em contribuir com métodos e abordagens, e não focaram na análise de efetividade. Desta forma, pouco se sabe sobre o quão realmente são efetivas as técnicas de *machine learning* para classificar compreensão de código baseado em dados de EEG. Esta oportunidade de pesquisa foi tratada no Capítulo 6.

(4) Reportar lições apreendidas sobre a experiencia de medir a carga cognitiva. Reportar lições apreendidas é uma prática essencial porque são reportadas soluções a partir de boas ou más experiencias dos pesquisadores ou de profissionais da indústria. Pesquisadores e profissionais são pessoas de interesse em relatórios de lições apreendidas porque elas fornecem métodos úteis de modo antecipado. De acordo com a Figura 10, existe a falta de lições apreendidas em praticamente todas as áreas identificadas neste trabalho (RQ5). Reportar lições apreendidas de experiencias passadas não é uma prática comum na área de engenharia de software.

A falta de lições apreendidas é também uma consequência da ausência de estudos de replicação. Nestes estudos, os autores geralmente escolhem um estudo anterior para repetir, validar os resultados, e até mesmo aplicar novos métodos. Porém, faltam diretrizes detalhadas para reportar estudos de replicação na área de engenharia de software (SIEGMUND; SIEGMUND; APEL, 2015). Esta falta de diretrizes implica que pesquisadores não sabem o que a academia espera como contribuição concreta a partir de estudos de replicação. Nos estudos de replicação, autores poderiam reportar algumas deficiências ou melhorias em relação a dispositivos, métricas, ou métodos em relação aos resultados atuais com a versão anterior da pesquisa. Outra solução possível é extrair e inferir lições apreendidas a partir de estudos de validação já produzidos na academia aplicando *Ground Theory*. As anotações geradas a partir da aplicação do *Ground Theory* poderiam ser enviadas em formato de formulários para os autores envolvidos no estudo para confirmar, adicionar, ou revisar as possíveis lições aprendidas deduzidas a partir do estudo. Produzir lições apreendidas não está no escopo deste trabalho.

(5) Desenvolver uma ferramenta para prestar suporte ao monitoramento de carga cognitiva em ambientes realísticos. Ferramentas são um dos mais visíveis e esperados resultados pela indústria de software. Este campo de pesquisa promete utilizar o nível de carga cognitiva como um preditor de problemas de qualidade de software, melhorando as condições de manter o software. Porém, pesquisadores produziram poucas ferramentas para este propósito. Ao invés disso, grande parte dos estudos primários buscaram prestar suporte a análises multimodais de medidas da carga cognitiva em estudos de laboratório. Outro exemplo é o trabalho sobre o sistema de gerenciamento de interrupção de fluxo de trabalho, isto é, *FlowLight*. Há o desafio de

produzir ferramentas para indicar a alta carga cognitiva dos desenvolvedores. Ferramentas na área de engenharia de software poderiam reduzir a carga cognitiva dos desenvolvedores através da recomendação de tarefas que demandam menos esforço mental. Consequentemente, medir a carga cognitiva dos desenvolvedores poderia reduzir problemas de qualidade no código fonte (MÜLLER; FRITZ, 2016), melhorar a compreensão de código (SIEGMUND, 2016), e reduzir a dificuldade dos desenvolvedores nas tarefas de programação (FRITZ et al., 2014).

Desta forma, pesquisadores poderiam utilizar o corpo de conhecimento fornecido nos estudos primários para desenvolver ferramentas para serem aplicadas em cenários realísticos da indústria de software. A primeira solução seria de propor e desenvolver um ambiente de desenvolvimento integrado (IDE). Pesquisadores poderiam propor um *plugin* para a IDE Eclipse para coletar informações dos dispositivos e integrar com classificadores existentes de problemas de qualidade de software, dificuldade em tarefas de programação, e níveis de expertise com o objetivo de testar o comportamento e efetividade dessa IDE para prestar suporte aos desenvolvedores. Desenvolver uma ferramenta não está no escopo deste trabalho.

3.4 Trabalhos relacionados ao estudo sistemático

Esta seção descreve trabalhos que estão associados a esse mapeamento sistemático. Estes trabalhos estão resumidos na Tabela 22. Em particular, a Tabela 22 sumariza os principais trabalhos relacionados especificando o título, a respectiva referência, o método de pesquisa utilizado, a quantidade de estudos que foram categorizados, os objetivos de pesquisa, se o artigo aplicou protocolo de busca, faixa de tempo dos estudos pesquisados, e as dimensões investigadas. Além disso, um identificador de trabalhos relacionados (RW) foi atribuído a cada trabalho relacionado. Cada trabalho relacionado é discutido de acordo com esses critérios resumidos na Tabela 22.

RW1 BLASCO et al. (2016) revisaram a literatura sobre sistemas de reconhecimento biométrico *wearable*. Os autores analisaram artigos publicados até o ano de 2016. Esses artigos não foram selecionados seguindo um protocolo de busca. Os autores concentraram a análise em estudos que aplicaram e propuseram o uso de dispositivos biométricos *wearable* para sistemas de reconhecimento. Esses sistemas foram categorizados de acordo com os tipos de sensores vestíveis, as técnicas de aprendizado de máquina aplicadas ao processamento dos sinais desses dispositivos, e as métricas para quantificar esses sinais.

RW2 FIGL (2017) revisou a literatura sobre a compreensão de modelos de processo de negócios. Os autores analisaram 79 artigos para realizar a revisão da literatura. Estes artigos foram selecionados usando um protocolo de busca. Os autores analisaram artigos publicados até o ano 2016. Esses artigos foram classificados e discutidos em relação a fatores que impactam na carga cognitiva dos usuários, variáveis independentes e dependentes associadas à compreensão do modelo de processo de negócio, revisão de variáveis que causam aumento da carga cognitiva e, portanto, que impactam na compreensão desses modelos, e como os modelos de processos

de negócios são compreendidos de acordo com diferentes níveis de experiência.

RW3 GUI et al. (2019) realizaram uma revisão dos estudos que aplicam métricas relacionadas as ondas cerebrais para fins de identificação e autenticação dos usuários. Os autores pesquisaram cerca de 188 artigos para discutir sobre várias dimensões relacionadas a essas métricas. Eles discutiram dimensões, tais como, os dispositivos de EEG mais utilizados para coletar dados cerebrais, o tipo de tarefas experimentais usadas para avaliar sinais cerebrais, onde obter conjunto de dados públicos de EEG, técnicas para pré-processamento e pós-processamento de sinais cerebrais, técnicas e diretrizes para os recursos. Listaram técnicas de *machine learning* para extração de *features* relacionadas a sinais de EEG, e técnicas para classificar esses dados. Além disso, discutem questões de segurança e estabilidade da aplicação da biometria cerebral em cenários reais.

RW4 BABLANI et al. (2019) analisaram estudos relacionados as Interfaces Cérebro Computador (BCI) aplicadas em um contexto interdisciplinar. Os autores analisaram cerca de 188 artigos para responder questões relacionadas à aquisição e processamento de sinais dos BCIs. Os artigos analisados abrangem estudos publicados até o ano de 2018. BABLANI et al. (2019) não aplicaram nenhum protocolo de busca para a seleção dos estudos. BABLANI et al. (2019) analisaram dimensões tais como, as categorias de aquisição de sinais, bem como os respectivos dispositivos para cada dessas categorias, os tipos de sinais cerebrais, os recursos que podem ser extraídos dos sinais cerebrais e as técnicas de classificação.

RW5 WILBANKS; MCMULLAN (2018) conduziu uma revisão da literatura sobre a carga cognitiva de enfermeiros durante o uso do *Electronic Health Records*. Os autores argumentaram que índices mais altos de carga de trabalho cognitiva implicam em erros de enfermeiros. Os autores aplicaram um protocolo de pesquisa para selecionar sistematicamente os estudos, ou seja, escolheram os mecanismos de busca específicos, construíram uma *string* de busca e utilizaram um processo para incluir e excluir estudos. Após esse processo, eles selecionaram 14 estudos para investigar as questões de pesquisa. Os trabalhos selecionados abrangem um intervalo entre os anos de 2010 e 2017. Os autores analisaram esses trabalhos para investigar dimensões como medidas de carga cognitiva, tamanho da amostra dos dados obtidos nos experimentos, e direcionamentos futuros.

RW6 LOTTE et al. (2007) analisou estudos sobre os algoritmos de *machine learning* de classificação e regressão para dispositivos sinais EEG. Os autores não utilizaram um protocolo de busca para a seleção dos artigos primários. Os artigos abrangem um período de publicação entre os anos 2007 e 2017. Esta revisão da literatura analisou cerca de 249 artigos. Os autores analisaram esses artigos para sumarizar quais algoritmos de classificação são ideais para os sinais do eletroencefalograma. Por isso, esse trabalho investigou quais técnicas de extração de *features* são utilizadas, listou quais técnicas de *machine learning* são utilizadas para predizer e classificar resultados baseados em sinais do eletroencefalograma, bem como, onde e qual situação utilizar essas técnicas.

RW7 MOHANANI et al. (2018) conduziram um estudo de mapeamento sistemático sobre

Tabela 22: Principais características dos trabalhos relacionados.

ID	Título	Método de Pesquisa	# Estudos	Protocolo de Busca	Domínio	Aim(s)	Período	Questões/ Aspectos investigados
RW01	A Survey of Wearable Biometric Sistemas de reconhecimento (BLASCO et al., 2016)	Survey	Não especifica	Não	Reconhecimento Biométrico	Explorar questões específicas sobre sistemas de reconhecimento biométrico.	Until 2016	Classificação de sensores wearables, técnicas para processar sinais brutos, técnicas de aprendizado de máquina e discussão de questões tais como a filtragem de sinais biológicos, e falta de reconhecimento biométrico conjuntos de dados.
RW02	Comprehension of Procedural Visual Business Process Models A Literature Review (FIGL, 2017)	Revisão da Literatura	79	Sim	Compreensibilidade de gráfico de fluxos	Categorize os fatores que influenciam a carga cognitiva na compreensão dos modelos de processos de negócios.	Até 2016	Estrutura para fatores que influenciam a carga cognitiva, Variáveis relacionadas ao processo de compreensão, visão geral de variáveis independentes que são fontes de carga cognitiva e impactos na compreensão de modelos, perspectiva da compreensão de modelos de processo, discussão de lacunas de pesquisa.
RW03	A Survey on Brain Biometrics (GUI et al., 2019)	Survey	∼188	Sim	Reconhecimento Biométrico	Descrever as pesquisas científicas atuais sobre métricas cerebrais destinadas a sistemas de identificação e autenticação.	2007 a 2017	Dispositivos EEG, tarefas, bancos de dados públicos de EEG, técnicas de processamento de dados de EEG técnicas de classificação, e desafios de pesquisa.
RW04	Survey on Brain-Computer Interface: An Emerging Computational Intelligence (BABLANI et al., 2019)	Survey	181	Não	Interfaces Cérebro-Computador	Fornecer uma sumarização sobre técnicas envolvendo ECoG, EEG, MEG, e MRI.	Até 2018	Tipos de aquisição de sinais cerebrais, técnicas para extração de recursos e algoritmos de classificação.
RW05	A Review of Measuring the Cognitive Workload of Electronic Health Records (WILBANKS; MCMULLAN, 2018)	Literature Review	14	Yes	Health	Summarize literature on measuring the cognitive workload of clinicians during the utilization of the Electronic Health Records.	2010 to 2017	Measures, population, sample size, and findings.
RW06	A review of classification algorithms for EEG-based brain-computer interfaces: A 10-year Update (LOTTE et al., 2007)	Literature Review	249	Não	Reconhecimento de Padrões	Sumarização de algoritmos de classificação	2007 a 2017	Seleção de MLT Features, algoritmos de classificação e regressão.
RW07	Cognitive Biases in Software Engineering: A Systematic Mapping Study (MOHANANI et al., 2018)	Mapeamento Sistemático	65	Sim	Viés Cognitivo	Classifica e categoriza estudos sobre vieses cognitivos em engenharia de software.	1990 and 2016	Tipo de viés cognitivo, as causas do viés cognitivo, os efeitos do viés cognitivo, abordagens para *debiasing*, método de pesquisa, ano, fonte da publicação, área de conhecimento.
RW08	Categorisation of Mobile EEG: A Researcher?s Perspective (BATESON et al., 2017)	Revisão da Literatura	29	Sim	Interface Cérebro-Computador	Define categorias para dispositivos móveis ou EEG ambulatoriais.	Até 2017	Sumarização de um esquema para classificar dispositivos EEG , discussões de resultados e deficiências no esquema para classificação.
RW09	Eye tracking in library and information science: a literature review (LUND, 2016)	Revisão Sistemática da Literatura	59	Sim	Métricas de Eye Tracker	Classificar os estudos que aplicam a tecnologia de *Eye Tracker* no campo de sistemas de informação.	2005 a 2015	Dispositivos de *Eye Tracker*, medidas de *Eye Tracker*, método para análise de dados e métodos para coletar dados de dispositivos de rastreamento ocular.
RW10	Psychophysiological measures of human cognitive states applied in human computer interaction (DIRICAN; GÖKTÜRK, 2011)	Revisão da Literatura	∼20	Sim	Interação Humano-Computador	Revisar e classificar medidas psicofisiológicas de estados cognitivos aplicados na Interação Humano-Computador.	Até 2010	Medidas de estados cognitivos, e as possíveis aplicações práticas.

Fonte: Elaborado pelo autor.

vieses cognitivos na engenharia de software. Os autores abordam o viés cognitivo como a propensão dos desenvolvedores de software a impor seus pontos de vista ou crenças nas decisões durante tarefas de engenharia de software. Os autores filtraram sistematicamente os estudos primários por meio de um protocolo de busca. Para isso, os autores selecionaram 65 artigos a partir de 826 estudos. Esses estudos foram publicados entre os anos de 1990 e 2016. MOHANANI et al. (2018) analisaram e classificaram algumas dimensões em relação ao viés cognitivo, tais como, origem do viés cognitivo, consequências no ambiente, métodos para tratar o viés cognitivo, métodos de pesquisa utilizados pela literatura, locais onde esses estudos foram publicados, e área de conhecimento de engenharia de software para a qual esse problema foi investigado.

RW8 BATESON et al. (2017) revisaram a literatura para definir como classificar os dispositivos EEG sem fio de acordo com o esquema proposto. Existem vários tipos de EEG. Os autores propuseram a classificação dos dispositivos de EEG com base na taxa de amostragem, número de canais, tipo de eletrodo e duração da bateria. Dispositivos como o NeuroSky (KATONA et al., 2014) e Emotiv (EMOTIV, 2014), foram amplamente utilizados em experimentos de engenharia de software, foram classificados como dispositivos com baixa precisão temporal em relação aos dispositivos de EEG utilizados em clínicas médicas de portabilidade limitada.

RW9 LUND (2016) conduziu uma revisão sistemática da literatura sobre rastreamento ocular no campo de pesquisa em bibliotecas e ciência da informação. Os autores analisaram 59 estudos os quais foram publicados entre 2005 e 2015 e foram obtidos a partir de pesquisa em dois mecanismos de pesquisa. Os autores categorizaram os estudos primários selecionados de acordo com tipos de métricas, métodos de pesquisa, locais de pesquisa e número de participantes.

RW10 DIRICAN; GÖKTÜRK (2011) revisaram a literatura das métricas psicofisiológicas aplicadas à interação homem-computador (HCI). As métricas de HCI servem para avaliar a experiência dos usuários em relação à interação com os computadores. Os autores investigaram 20 estudos publicados até o ano de 2010. Eles classificaram os estudos com base nas medidas utilizadas em HCI, e onde e como essas medidas psicofisiológicas foram aplicadas.

Em resumo, este capítulo realiza um mapeamento sistemático sobre as medidas de carga cognitiva no domínio de engenharia de software. Para isso, adotamos um protocolo de busca, e foi mapeado sistematicamente 63 estudos primários publicados até maio de 2020 sobre carga cognitiva na engenharia de software. Finalmente, a análise contida neste estudo contribui para (a) fornecer um entendimento sobre o estado da arte em relação as medidas de carga cognitiva dos desenvolvedores, e (b) identificar questões em aberto e oportunidades futuras.

3.5 Ameaças a validade

Esta Seção apresenta procedimentos para lidar com ameaças à validade. Especificamente, essas ameaças foram tratadas em relação à validade interna, de construção, e de conclusão (WOHLIN et al., 2012).

Ameaças de construção. Este tratamento atenua as ameaças em relação aos construtores usados neste trabalho os quais podem influenciar no processo de seleção dos estudos primários. Algumas diretrizes foram seguidas para selecionar estudos primários que são estritamente relacionados ao objetivo de pesquisa. Primeiro, os principais termos e sinônimos que formam a *string* de busca foram extraídos das *keywords* de artigos que são referências principais a este tópico de pesquisa. Essas *keywords* formaram uma sequência de sinônimos e palavras equivalentes denominada de *string* de busca, a qual também foi concebida com base em práticas bem definidas na literatura (KITCHENHAM; CHARTERS, 2007; KITCHENHAM; BUDGEN; PEARL BRERETON, 2011). Essa *string* de busca foi usada nos principais mecanismos de busca relacionados a área de pesquisa.

Validade interna. Este aspecto de validade trata sobre os cuidados para evitar que fatores internos não afetem os resultados, ou seja, se os resultados foram realmente derivados de uma classificação confiável. Para isso, a classificação dos artigos foi realizada de acordo com os termos existentes no domínio de engenharia de software, bem como a literatura de carga cognitiva relacionada a engenharia de software (BRAISBY; GELLATLY, 2012). Além disso, a análise e os resultados foram analisados pelo autor. O autor também revisou cada etapa do processo de seleção, bem como a classificação dos estudos primários em relação a cada questão de pesquisa.

Validade de conclusão. Trata-se de medidas tomadas para evitar conclusões enviesadas a partir dos resultados obtidos. Em particular, foi evitado o problema de *fishing error*, onde busca-se analisar os dados procurando uma conclusão previamente arquitetada, ou seja, as conclusões não foram delineadas antes da análise os resultados. Por isso, buscou-se delinear conclusões somente após a coleta de todos os resultados (WOHLIN et al., 2012). Por fim, todos os resultados e análises foram realizados com base na literatura de mapeamento sistemático.

3.6 Lista de trabalhos relacionados à abordagem CogEff

A lista de trabalhos relacionados à técnica, foi selecionada baseada nos seguintes critérios: (1) eram estudos relacionados à compreensão de programas, (2) buscavam entender quais regiões cerebrais estavam relacionadas a compreensão de programas, e (3) buscavam adaptar abordagens para medir a carga cognitiva em tarefas de engenharia de software. Nesta Seção também foram adicionadas três principais estudos relacionados que não estão inclusos no processo de seleção, pois este processo considerou estudos até maio de 2020.

HIJAZI et al. (2022) mediram a taxa de batimento cardíaco (HRV), e dados pupilométricos de desenvolvedores para medir a carga cognitiva durante tarefas de revisão de código. HIJAZI et al. (2022) utilizaram dados do HRV e pupilométricos, complexidade de código, tempo de análise do código fonte, e quantidade de revisões para classificar a qualidade da revisão do código fonte. Com base nesses dados, HIJAZI et al. (2022) desenvolveram uma abordagem baseada em regras e conhecimento de especilistas para definir a situação em que resultam em uma boa ou má revisão de código. Esta abordagem apresentou uma eficácia de 84.6% para classificar a

revisão de código. Além disso, HIJAZI et al. (2022) também utilizaram a combinação de dados psicofisiológicos e dados do código fonte em ambas as técnicas de *machine learning* K-Nearest Neighbors (KNN) e regressão logística (RL) para realizar uma classificar a revisão de código de desenvolvedores (revisão boa/ruim). Os resultados mostram que o classificador KNN teve uma efetividade de 84.8% ao classificar que desenvolvedores realizaram uma má revisão, sem que todos os bugs fossem detectados. Além disso, o classificador de regressão logística (RL) atingiu uma eficácia de 86.5% para classificar que desenvolvedores realizaram uma má revisão, sem que todos os bugs fossem detectados. Por fim HIJAZI et al. (2022) também utilizaram dados de EEG para validar os resultados obtidos dos classificadores treinados com dados de taxa de batimentos cardíacos e pupilometria (ANS). Para isso, HIJAZI et al. (2022) treinaram as técnicas KNN e regressão logística com ondas theta (θ), beta (β) e gamma (γ). HIJAZI et al. (2022) constataram que a precisão, recall, e acurácia do classificador treinado com dados EEG foi similar ao classificar treinados com dados de batimentos cardíacos e pupilometria. De acordo com os autores, a eficácia dos classificadores treinados com dados de EEG valida os resultados dos classificadores treinados com dados de ANS, devido a relação do EEG com a carga cognitiva.

PEITEK et al. (2021) coletaram dados BOLD de desenvolvedores partir de fMRI enquanto eles executavam tarefas de compreensão. A partir desses dados, PEITEK et al. (2021) monitoraram dados relacionados as áreas cerebrais ativas durante a compreensão de código. PEITEK et al. (2021) também analisaram a correlação dos dados *Blood Oxygen Level Dependent* (BOLD) obtidos do fMRI com métricas de software. A análise de PEITEK et al. (2021) buscou encontrar uma correlação das métricas de software relacionadas a aspectos de tamanho do código, complexidade, e tamanho de vocabulário do código fonte. Em relação as áreas cerebrais, PEITEK et al. (2021) encontraram atividade nas áreas de Broadman (BA) 6, BA 21, BA 39, BA 44 e BA 45. Os resultados apresentaram que a complexidade de McCabe não teve correlação com nenhuma das áreas cerebrais ativas durante as tarefas de compreensão de código. Porém, o aspecto de tamanho, através da quantidade de linhas apresentou correlação com a BA 21 e uma correlação baixa com as áreas BA 6, BA 39, BA 44 e BA 45. Além disso, aspectos do tamanho do código (quantidade de linhas) e variedade do vocabulário do código fonte (*Halstead*) apresentaram correlação positiva em relação a desativação nas áreas BA 31 e BA 32. Estes resultados são importantes pois relacionaram a atividade cerebral diretamente com aspectos relacionados à tarefas de compreensão de código.

KARAS et al. (2021) propuseram o uso de análise de conectividade funcional em dados de fMRI aplicados a área de engenharia de software. A conectividade funcional representa a atividade cerebral através de um grafo, onde cada nodo é uma região cerebral e a conectividade representa as arestas entre os diferentes nodos. KARAS et al. (2021) aplicaram essa análise com o objetivo de investigar a interação das regiões cerebrais em tarefas de compreensão de código e texto simples. A diferença entre os estudos de fMRI convencionais é considerar a relação entre a interação das regiões ao invés de apenas analisá-las de forma isolada. KARAS et al. (2021)

realizaram essa análise de conectividade funcional em um conjunto de dados já existente. Nessa base de dados continham dados BOLD que foram coletados de desenvolvedores durante tarefas de compreensão de código. Os principais resultados mostram que existe uma forte conexão entre as áreas cerebrais responsáveis pelo processamento semântico de linguagem natural. Além disso, outro resultado relevante de KARAS et al. (2021) foi uma significante diferença entre as conexões entre regiões cerebrais durante tarefas com linguagens diferentes, isto é, entre código fonte e tarefas de texto simples.

CRK; KLUTHE; STEFIK (2015) realizaram um experimento controlado utilizando ondas cerebrais obtidas dos desenvolvedores a partir de um eletroencefalograma. Foram extraídas as ondas alfa e theta do eletroencefalograma. Para medir o esforço mental, CRK; KLUTHE; STEFIK (2015) adotaram o indicador *Event Related Desyncronization* (ERD), pois, baseado na evidência da literatura encontrada por eles, altos valores de ERD são refletidos por uma demanda maior da utilização da memória de trabalho dos participantes. O objetivo do estudo foi relacionar o nível de experiência dos desenvolvedores ao nível da carga cognitiva. Os autores chegaram à conclusão de que o ERD é maior em participantes com menos experiência em relação a programação. A carga cognitiva foi utilizada para investigar o nível de experiência dos participantes. Analisou a correlação entre o ERD e as respostas das tarefas de compreensão. A análise foi realizada para verificar a relação entre o ERD e as respostas obtidas a partir das de compreensão de código. A análise não foi realizada para verificar a viabilidade de adotar o ERD como indicador de carga cognitiva em tarefas de compreensão. Estudo não focou em classificar compreensão de código utilizando técnicas de *machine learning*.

PEITEK et al. (2018a) coletaram *Blood Oxygen Level Dependent* (BOLD) através de um fMRI dos participantes durante tarefas de compreensão de código. Esta é uma abordagem que realça áreas de atividade cerebral durante tarefas de interesse. Foi investigado a ativação de áreas cerebrais durante as tarefas de compreensão de código. PEITEK et al. (2018a) analizaram a correlação dos dados BOLD com algumas métricas de código, tais como, complexidade de *Halstead*, complexidade ciclomática, e *DeepDegree*. PEITEK et al. (2018a) não encontraram correlação entre BVP e as métricas de software testadas. Além disso, a análise de correlação foi avaliada em relação a aspectos do código, e não aspectos psicométricos, tais como, as respostas das tarefas de compreensão, e o tempo para concluir a tarefa. PEITEK et al. (2018a) não focaram em classificar compreensão de código baseado em dados psicofisiológicos.

SIEGMUND et al. (2017) também como apresentado em PEITEK et al. (2018a) coletaram dados BOLD dos participantes durante tarefas de compreensão de código. Esses dados também foram obtidos do fMRI. Especificamente, SIEGMUND et al. (2017) verificaram se existia uma diferença no indicador BOLD entre tarefas de compreensão que continham códigos fonte com dicas semânticas e sem dicas semânticas. Os resultados mostraram que, os níveis dos valores BOLD foram significativamente menores em tarefas de compreensão de código que continham dicas semânticas. SIEGMUND et al. (2017) não focaram em classificar compreensão de código baseado em dados psicofisiológicos com técnicas de *machine learning*, nem avaliar a efetivi-

dade dessas técnicas.

DURAISINGAM; PALANIAPPAN; ANDREWS (2017) coletaram ondas cerebrais dos participantes a partir do EEG para investigar o nível de dificuldade em tarefas cognitivas, tais como tarefas matemáticas, e raciocínio lógico. DURAISINGAM; PALANIAPPAN; ANDREWS (2017) utilizaram as bandas alfa (α), beta (β), delta (δ), gama (γ), e theta (θ) do EEG. A partir desses dados, DURAISINGAM; PALANIAPPAN; ANDREWS (2017) classificaram a dificuldade dos participantes. Os resultados apontaram que a técnica *Naïve Bayes* treinada com dados EEG classificou dificuldade de tarefas dos participantes com efetividade superior a 80%. A pesquisa de DURAISINGAM; PALANIAPPAN; ANDREWS (2017) não propôs uma abordagem específica para mensurar a carga cognitiva para compreensão de código, e não avaliou a correlação dos indicadores utilizados com tarefas de compreensão de código.

LEE et al. (2017) capturam dados de ondas cerebrais obtidas de um EEG, e de um *Eye-Tracker* para investigar o nível de dificuldade das tarefas, e conhecimento dos desenvolvedores em tarefas de compreensão de código. A partir do EEG LEE et al. (2017) extraíram as ondas alfa (α), beta (β), delta (δ), gama (γ), e theta (θ). LEE et al. (2017) calcularam a potência de banda e então treinaram uma técnica de *machine learning Support Vector Machine* (SVM) para classificar nível de dificuldade, e outro SVM para classificar o nível de experiência dos desenvolvedores. Os resultados apontaram que a técnica obteve uma efetividade de 97% para classificar o nível de conhecimento, e uma efetividade de 64.9% para classificar o nível de dificuldade. LEE et al. (2017) correlacionaram o nível de experiência dos desenvolvedores com dados auto reportados pelos desenvolvedores em relação a seus conhecimentos individuais. Não desenvolveu abordagem específica para medir a carga cognitiva na compreensão de software.

KOSTI et al. (2018) utilizaram as ondas cerebrais a partir do EEG dos desenvolvedores para investigar nível de dificuldade de tarefas. KOSTI et al. (2018) coletou do EEG as bandas alfa (α), beta (β), delta (δ), gama (γ), e theta (θ). Os dados foram coletados enquanto desenvolvedores executavam tarefas de compreensão de código. KOSTI et al. (2018) analisaram a correlação entre os dados de ondas cerebrais e a dificuldade auto reportada pelos participantes. O teste mostrou que as ondas cerebrais e os níveis de dificuldade estão correlacionados, principalmente em relação as ondas beta (β), e gama (γ). Não foi proposta uma abordagem específica para mensurar carga cognitiva em tarefas de compreensão de código. KOSTI et al. (2018) não classificaram a compreensão de código baseado nos dados de EEG.

FRITZ et al. (2014) utilizaram dispositivos de EEG, *Eye-Tracker*, e EEG, coletando assim dados multimodais com a finalidade de investigar a dificuldade em tarefas de programação dos desenvolvedores. Especificamente a partir do EEG, FRITZ et al. (2014) extraíram as bandas alfa (α), beta (β), delta (δ), gama (γ), e theta (θ) para treinar as técnicas *Support Vector Machine*, *Naïve Bayes* e *Random Forrest* para classificar categorias de dificuldade em tarefas de programação dos desenvolvedores. Os dados biométricos foram coletados enquanto desenvolvedores programavam em c#. Os desenvolvedores, também, em cada tarefa forneceram um ranking em uma escala de 20 pontos aspectos em relação a percepção deles sobre o nível de

dificuldade. FRITZ et al. (2014) analisaram a correlação entre os dados do questionário, com o nível de dificuldade que foi atribuído as tarefas. O resultado da correlação mostrou que existe correlação entre a percepção reportada pelos desenvolvedores e os níveis de dificuldade de tarefa. Os classificadores alcançaram uma efetividade acima de 85% para classificar dificuldade de tarefas de programação.

MÜLLER; FRITZ (2016) coletaram dados dos dispositivos de atividade eletrodérmica, taxa de respiração, e variação de batimentos cardíacos em técnicas dos participantes enquanto executavam tarefas de programação. MÜLLER; FRITZ (2016) utilizaram esses dados para treinar uma técnica de *machine learning* para classificar problemas de qualidade de software, tais como: estilo de código, *bugs*, e ausência de tratamento de exceções. MÜLLER; FRITZ (2016) concluíram que o maior nível de dificuldade dos desenvolvedores está correlacionado a uma maior quantidade de problemas de código. A técnica de *machine learning* apresentou uma efetividade acima de 43% para classificar problemas de qualidade de código baseado em dados psicofisiológicos. O trabalho de MÜLLER; FRITZ (2016) não utilizou EEG, nem mesmo avaliou correlação entre indicadores psicofisiológicos e tarefas de compreensão de código.

CASTELHANO et al. (2019) utilizaram um fMRI para extrair dados BOLD para analisar a atividade cerebral durante tarefas de compreensão de código. O principal objetivo do trabalho de CASTELHANO et al. (2019) era identificar atividades em áreas relacionadas a detecção dos *bugs* no código fonte pelos participantes. Para isso, CASTELHANO et al. (2019) coletaram dados dos desenvolvedores enquanto executavam tarefas de compreensão de código, onda algumas continham *bugs*. CASTELHANO et al. (2019) identificaram a ativação de áreas relacionadas a tomada de decisão e incerteza, processamento matemático, e coordenação de múltiplas tarefas. O estudo não tem foco em aplicar EEG, não propõe uma abordagem específica para mensurar a carga cognitiva em tarefas de compreensão de código, e não busca classificar compreensão de código baseado em dados psicofisiológicos.

FAKHOURY et al. (2018) coletaram dados BOLD dos desenvolvedores a partir de um dispositivo fNIRS. Os dados foram coletados enquanto desenvolvedores executavam tarefas de compreensão de código. O objetivo de FAKHOURY et al. (2018) foi de analisar os efeitos que a legibilidade e organização do código fonte causam no esforço mental dos desenvolvedores. Após o fim do experimento, FAKHOURY et al. (2018) aplicaram o teste estatístico para verificar a diferença entre o esforço aplicado para resolver tarefas classificadas em diferentes grupos de problemas de código. Os resultados mostraram que problemas estruturais e de legibilidade exigem uma demanda significativa da carga cognitiva dos desenvolvedores. O estudo de FAKHOURY et al. (2018) não busca analisar a correlação entre indicadores psicofisiológicos em relação a compreensão de código. Este estudo também não busca classificar compreensão de código baseado em dados EEG.

FUCCI et al. (2019) adotaram dispositivos de EEG, batimentos cardíacos, e atividade eletrodérmica para investigar se é possível classificar o tipo de tarefa (texto ou código) em relação a dados psicofisiológicos. Neste caso, desenvolvedores realizaram tarefas de leitura textual, e de

compreensão de programas. FUCCI et al. (2019) extraíram as ondas alfa (α), beta (β), delta (δ), theta (θ), e gama (γ). A partir desses dados, FUCCI et al. (2019) treinaram classificadores, tais como, *Support Vector Machine*, e uma *Neural Network*. O classificador *Neural Network* atingiu uma efetividade de 70% para classificar o tipo de tarefa. Os autores correlacionaram o resultado do classificador com a nota média de curso dos participantes para analisar se a classificação tem correlação com o nível de conhecimento dos participantes. A associação foi calculada através do coeficiente de *Kendall*, o qual demonstrou pouca correlação entre expertise e a acurácia dos classificadores. Estudo não propôs ou desenvolveu uma abordagem específica para mensurar carga cognitiva em tarefas de compreensão de código.

YEH et al. (2017) utilizaram um dispositivo de EEG para investigar a atividade cerebral durante tarefas de compreensão de código. YEH et al. (2017) coletaram dados da onda cerebrais dos desenvolvedores enquanto realizavam tarefas de compreensão de código. A partir do EEG, YEH et al. (2017) extraíram as ondas alfa (α) e theta (θ) com o objetivo de investigar a magnitude das ondas alfa (α) e theta (θ) entre tarefas com código confuso, em relação a tarefas de código não confuso. Para testar essa diferença, YEH et al. (2017) aplicaram o *t-test* entre esses dois grupos de tarefas e os resultados apontaram que a diferença era significativa nas ondas alfa (α) e theta (θ) entre tarefas de código confuso em relação a código não confuso. YEH et al. (2017) analisaram a correlação de entre a potência absoluta da onda alfa com o número de respostas corretas dos participantes. O resultado da correlação apresentou que existe correlação entre onda alfa (α) e a performance do desenvolvedor. Este trabalho não desenvolveu uma abordagem especifica para mensurar carga cognitiva em tarefas de compreensão, e também não classificou compreensão de código baseado em dados psicofisiológicos.

3.7 Análise Comparativa

A partir dos estudos selecionados, uma análise comparativa foi realizada para determinar as principais lacunas de pesquisa a qual esta tese pretende resolver. A Tabela abaixo realiza uma análise comparativa em relação aos seguintes critérios:

- **C1. EEG:** coleta ondas cerebrais a partir do EEG;
- **C2. Carga cognitiva:** mede a carga cognitiva dos desenvolvedores em tarefas de compreensão de código;
- **C3. Abordagem para mensurar carga cognitiva em tarefas de compreensão de código:** propõe uma abordagem para mensurar a carga cognitiva em tarefas de compreensão de código;
- **C4. Análise Correlação:** avalia a correlação de indicadores de carga cognitiva com tarefas de compreensão de código;

- **C5. Classificação de compreensão de código baseado em indicadores psicofisiológicos dos desenvolvedores:** realiza treinamento de técnicas de *machine learning* com dados psicofisiológicos para classificar compreensão de código.

- **C6. Analisa a efetividade de técnicas de *machine learning* para classificar compreensão de código:** utiliza métricas tais como precisão e *recall*, para avaliar a efetividade de treinar técnicas de *machine learning* treinadas com dados EEG para classificar a compreensão de código dos desenvolvedores.

Tabela 23: Lacunas dos trabalhos relacionados.

Trabalhos	C1. EEG					C2. Carga Cognitiva	C3. Abordagem Específica	C4. Análise de correlação	C5. Classificação Compreensão de Software	C6. Estudo Empírico avaliação efetividade
	Alfa .	Beta .	Delta .	Gama .	Theta .					
CogEff	+	+	+	+	+	+	+	+	+	+
HIJAZI et al. (2022)	-	+	-	+	+	+	-	∼	∼	∼
PEITEK et al. (2021)	-	-	-	-	-	+	-	∼	-	-
KARAS et al. (2021)	-	-	-	-	-	+	-	∼	-	-
CRK; KLUTHE; STEFIK (2015)	+	-	-	-	+	∼	-	∼	-	-
PEITEK et al. (2018a)	-	-	-	-	-	+	-	∼	-	-
SIEGMUND et al. (2017)	-	-	-	-	-	+	-	-	-	-
DURAISINGAM; PALANIAPPAN; ANDREWS (2017)	+	+	+	+	+	∼	-	-	∼	∼
LEE et al. (2017)	+	+	-	+	+	∼	-	∼	∼	∼
KOSTI et al. (2018)	+	+	+	+	+	+	-	∼	-	-
FRITZ et al. (2014)	+	+	+	+	+	∼	-	∼	∼	∼
MÜLLER; FRITZ (2016)	-	-	-	-	-	∼	-	-	∼	∼
CASTELHANO et al. (2019)	-	-	-	-	-	∼	-	-	-	-
FAKHOURY et al. (2018)	-	-	-	-	-	∼	-	-	-	-
FUCCI et al. (2019)	+	+	+	+	+	∼	-	∼	∼	∼
YEH et al. (2017)	+	-	-	-	+	∼	-	∼	∼	∼

Legenda: + Contém - Não Contém ∼ Contém Parcialmente

Fonte: Elaborado pelo autor.

A partir da Tabela 23 é possível identificar algumas oportunidades de pesquisa:

(1) Desenvolvimento de uma abordagem para medir a carga cognitiva em tarefas de compreensão de código: A Tabela 23 evidencia a falta de uma abordagem específica para medir a carga cognitiva em tarefas de compreensão de software. Primeiro, no critério C1 apresenta uma discordância de quais ondas cerebrais considerar em uma abordagem para mensurar a carga cognitiva na engenharia de software, e por isso, nem todas essas ondas cerebrais foram utilizadas nos trabalhos que adotaram EEG. É importante considerar todas, já que, elas têm papel importante na carga cognitiva. Segundo, porque de acordo com o critério C2, grande parte dos estudos não mediram a carga cognitiva diretamente em tarefas de compreensão de código, ao invés disso, grande parte dos trabalhos focaram em medir a carga cognitiva em propósitos, tais como, dificuldade de tarefas e nível de experiencia dos desenvolvedores. Por último, porque de acordo com o critério C3 na Tabela 23, nenhum dos principais estudos propôs ou desenvolveu uma abordagem específica para medir a carga cognitiva nas tarefas de compreensão de código, ao invés disso, houve a adaptação de abordagens para medir a carga cognitiva. A abordagem proposta encontra-se no Capítulo 4.

(2) Análise de correlação entre a abordagem desenvolvida com tarefas de compreensão de código: De acordo com a Tabela 23, também faltam análises sobre a correlação das abordagens para mensurar carga cognitiva com tarefas de compreensão de código. Essa análise é importante principalmente por causa de dois fatores envolvendo o critério C4 da Tabela 23. Primeiro, porque os estudos que analisaram a correlação, executaram em relação à propósitos relacionados a compreensão de código, tais como dificuldade de tarefas, ou com a experiência do desenvolvedor, sendo na prática uma avaliação parcial da correlação com tarefas de compreensão de código. Segundo porque, um pequeno grupo de estudos ignorou a análise da correlação com tarefas de compreensão de código. A falta dessas análises de correlação indica que existe o risco de adoção a adaptação de abordagens que talvez não sejam apropriadas em estudos envolvendo compreensão de código. A falta dessas análises reforça a necessidade da oportunidade 1. Dessa forma, consequentemente há a oportunidade de, principalmente, avaliar a abordagem desenvolvida. Essa análise visa principalmente avaliar a correlação da abordagem proposta com tarefas de compreensão de código. A análise foi realizada no Capítulo 5.

(3) Avaliação da efetividade de técnicas de *machine learning* para classificar compreensão de código baseado em dados EEG: Por fim, a Tabela 23 também evidencia a oportunidade de treinar técnicas de *machine learning* para classificar compreensão de código baseados em dados de EEG. Essa oportunidade se justifica por dois principais motivos. Primeiro, em relação ao critério C5, apenas alguns trabalhos atendem parcialmente esse critério, pois alguns trabalhos treinaram técnicas de *machine learning* com EEG, porém em propósitos diferentes de compreensão de código, mas em classificar o tipo de tarefas, nível de dificuldade, e do nível de experiência dos desenvolvedores. Além disso, existiu alguns estudos que não implementaram técnicas de *machine learning*. O segundo motivo que justifica essa oportunidade, é que, em relação ao critério C6, poucos estudos avaliaram a efetividade das técnicas de *machine learning* treinadas com dados EEG. Além disso, os que avaliaram a efetividade dessas técnicas, o fizeram de forma parcial, pois não utilizaram testes estatísticas para verificar se a efetividade das técnicas é maior que uma média considerada de alta performance. A avaliação da efetividade das técnicas de *machine learning* que classificam compreensão de código baseados em dados EEG foi realizada no Capítulo 6.

3.8 Considerações finais

Este capítulo realizou um mapeamento sistemático sobre medidas de carga cognitiva em engenharia de software. Para isso, foram selecionados 63 estudos primários dentre 4,175 estudos potencialmente relevantes. Os resultados da classificação desses estudos primários foram tendências e lacunas na pesquisa de carga cognitiva em engenharia de software. Além disso, essa classificação também gerou um esquema de classificação para estudos sobre carga cognitiva na engenharia de software. A seguir, esta pesquisa discutiu os resultados e esboçou os desafios futuros identificados nos estudos primários.

Este estudo realizou uma análise de dez questões de pesquisa. As principais conclusões de acordo com as respectivas questões de pesquisa são descritas a seguir: (1) Carga cognitiva: grande parte dos estudos tendem a não definir o conceito de carga cognitiva; (2) Sensores: a maioria dos estudos primários preferia combinar sensores para medir a carga cognitiva dos desenvolvedores. Os estudos primários que contaram com um sensor adotaram o *Eye-Tracker*, o eletroencefalograma ou a fMRI. (3) Métricas: estudos primários tendem a usar mais de uma métrica para medir a carga cognitiva. Como muitos trabalhos combinaram sensores os trabalhos aplicaram várias métricas para medir a carga cognitiva. Isso indica que muitas métricas estão relacionadas a carga cognitiva e podem melhorar a correlação da carga cognitiva com as tarefas de engenharia de software. (4) Técnicas de *machine learning*: os estudos primários tendem a adotar algoritmos de classificação, como *Support Vector Machines* (SVM) e *Naïve Bayes*. Uma parte dos trabalhos combinaram várias técnicas de aprendizado de máquina para analisar seu desempenho. No entanto, nenhum dos trabalhos combinou as técnicas de classificação com o objetivo de unir seus resultados para melhorar a precisão da classificação. (5) propósito: estudos primários focaram em investigar processos cognitivos centrais para o desenvolvimento de software, tais como compreensibilidade do código e dificuldade da tarefa; (6) tarefas: estudos focaram a investigação da carga cognitiva durante atividades de programação de código, preferencialmente tarefas de compreensão de código; (7) artefatos: o artefato preferido foi a utilização de trechos de código em Java nas tarefas de engenharia de software; (8) quantidade de participantes: pesquisadores em geral recrutaram entre 11 a 20 participantes. O recrutamento de participantes é difícil, principalmente devido à natureza do experimento variar, exigindo um perfil específico de participantes; (9) métodos de pesquisa: a maioria dos estudos primários em engenharia de software consistiam em estudos de validação; (10) locais de pesquisa: o número de publicações está em constante crescimento, atingindo o pico de publicações em 2019.

Baseado nesses resultados foram encontradas algumas lacunas a qual esta tese investiga, isto é, a falta de uma abordagem para medir a carga cognitiva de desenvolvedores em tarefas de compreensão de código, desconhecimento sobre a correlação dos indicadores psicofisiológicos com tarefas de compreensão de código, e falta de conhecimento empírico sobre a efetividade de técnicas de machine learning para classificar compreensão de código baseado em dados EEG. A investigação destas lacunas foi realizada nos próximos capítulos. Em particular, o próximo capítulo descreve o desenvolvimento da CogEff, uma abordagem para mensurar carga cognitiva de desenvolvedores de software em tarefas de compreensão de código.

4 COGEFF: UMA ABORDAGEM PARA MENSURAR A CARGA COGNITIVA EM TAREFAS DE COMPREENSÃO DE CÓDIGO

Este Capítulo responde à questão de pesquisa 2 (QP2), "Como propor uma abordagem baseada em dados EEG para mensurar esforço cognitivo durante tarefas de compreensão de código?". Essa questão visa resolver o problema da falta de uma abordagem específica de EEG para medir a carga cognitiva para a área de engenharia de software. Este problema foi notado ao perceber que, na área de engenharia de software, abordagens para medir a carga cognitiva foram geralmente adaptadas de outras áreas. Em particular, foi desenvolvido uma abordagem para medir a carga cognitiva em tarefas de compreensão de código, pois a compreensão de código é um propósito chave para o entendimento dos demais processos cognitivos na engenharia de software, tais como o nível de dificuldade, e de experiência dos desenvolvedores. Essa abordagem busca implementar as lacunas identificadas nas abordagens atuais tais como: a falta de considerar a conectividade entre diferentes canais de EEG, e a falta de propriedades para tornar a abordagem específica para tarefas de compreensão de código. Por isso, esta pesquisa propõe a CogEff, a qual trata-se de uma abordagem que utiliza dados de EEG para medir a carga cognitiva de desenvolvedores em tarefas de compreensão de código.

A Seção 4.1 apresenta a visão geral da abordagem CogEff, e como essa abordagem funciona. A Seção 4.2 apresenta os instrumentos utilizados para medir a carga cognitiva dos desenvolvedores. A Seção 4.3 apresenta o tipo de tarefa em que a CogEff mensura a carga cognitiva, e como essas tarefas são executadas pelos desenvolvedores. A seção 4.4 propões escalas para os resultados da CogEff. A Seção 4.5 descreve os principais pseudocódigos utilizados para executar a CogEff. A Seção 4.6 ilustra uma demonstração da execução da abordagem CogEff. Por fim, a Seção 4.7 apresenta uma análise comparativa das principais abordagens de EEG utilizadas para mensurar a carga cognitiva na engenharia de software. Por fim, a Seção 4.8 apresenta as considerações finais do capítulo.

4.1 Visão Geral da CogEff

Esta seção apresenta a visão geral da abordagem proposta, detalhando suas etapas e procedimentos para mensurar a carga cognitiva de desenvolvedores em tarefas de compreensão de código. Para isso, a Figura 11 apresenta uma visão geral do processo proposto da abordagem CogEff.

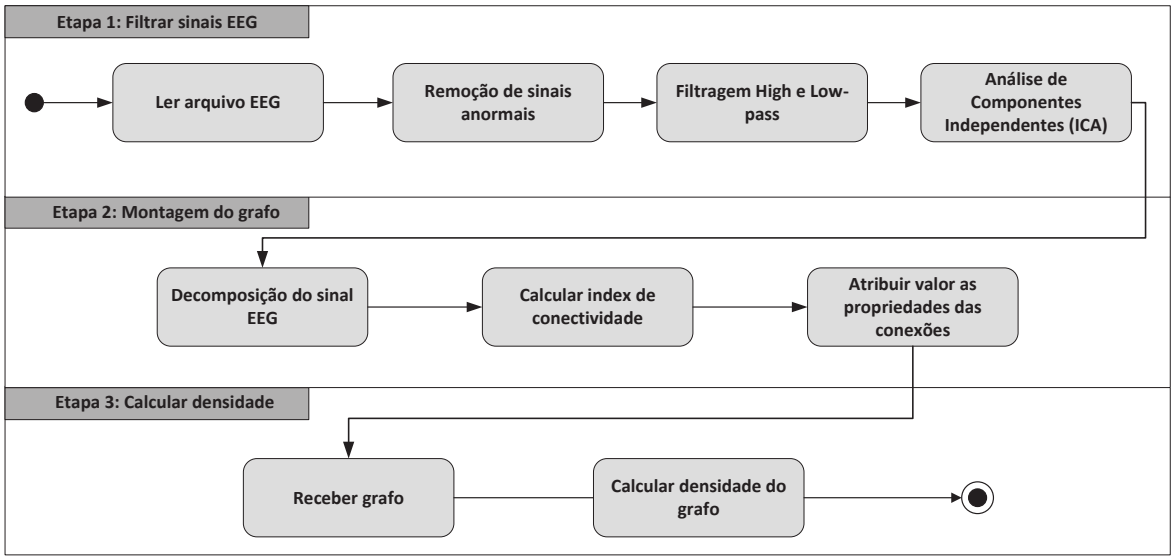
A CogEff foi desenvolvida para resolver o problema causado pela falta de uma abordagem específica para mensurar carga cognitiva na engenharia de software em tarefas de compreensão de programas. Especificamente, as abordagens atuais utilizadas na engenharia de software não possuem: (1) um processo para mesurar a carga cognitiva dos desenvolvedores na engenharia de software em tarefas de compreensão de código; (2) não propõem um processo para filtrar sinais dos canais EEG para tarefas de compreensão de código; (3) não definem propriedades

para tornar a abordagem específica para tarefas de compreensão de código; (4) não consideram a conectividade entre sensores utilizando uma abordagem *over-time*, e (5) não possuem um valor que resuma a carga cognitiva em tarefas de compreensão de código.

Por causa dessas lacunas a abordagem CogEff é composta por: (1) um processo para mesurar a carga cognitiva dos desenvolvedores de software em tarefas de compreensão de código, representado na Figura 11; (2) um processo que foi definido para filtrar sinais dos canais EEG em tarefas de compreensão de código, o qual consiste na Etapa 1 da abordagem; (3) uma abordagem para estabelecer conexões funcionais entre os canais EEG para considerar a interação entre esses canais, e sem depender de múltiplas *trials*, o qual consiste na Etapa 2; (4) torna a abordagem específica para compreensão de código pois define propriedades para atribuir pesos conforme a relevância de cada conexão a tarefa de compreensão de código, o qual consiste em parte da Etapa 2; (5) cálculo de densidade para resumir a conectividade do grafo obtido, o qual consiste na Etapa 3.

Especificamente, a Figura 11 apresenta que a abordagem CogEff é composta por três etapas para mensurar a carga cognitiva dos desenvolvedores: (1) filtrar sinais EEG, tem o objetivo de remover ruídos de cada um dos sinais de cada canal do EEG; (2) criar grafo, com o propósito de estabelecer conexões entre cada canal do EEG, gerando assim um grafo representando a conectividade entre os canais, e por fim, a etapa (3) calcular a densidade do grafo, o qual calcula uma densidade ponderada do grafo obtido da etapa anterior, e resulta um valor que representa a conectividade desse grafo.

Figura 11: Visão geral do processo proposto para mensurar carga cognitiva.



Fonte: Elaborado pelo autor.

4.1.1 Etapa 1: Filtrar sinais EEG

Esta Etapa tem como objetivo filtrar os sinais de EEG. Esta seção apresenta o *design* do filtro de EEG, junto a outras medidas para atenuar ruídos e inconsistências do sinal, tais como, a análise em componentes independentes (ICA) para diminuir a quantidade de artefatos oculares no sinal de EEG.

Remoção de sinais anormais. Conforme a Figura 11, o primeiro passo da etapa de filtragem, trata-se de descartar sinais anormais que estejam no intervalo de acima de 8.000μ, e abaixo de -8.000μ. Esses valores são então removidos e não são substituídos por outros valores. Esses valores foram obtidos após observar a presença de leituras anormais pelos sensores, que se enquadravam especificamente nessa faixa de valor.

Filtro *High-Low pass*. A Figura 11 mostra que após a remoção de sinais anormais contidos no EEG, esses sinais passam pelo filtro *High* e *low-pass*. Apesar de não livrar o sinal EEG de todas as interferências, esse filtro é importante para atenuar os efeitos dos ruido nas faixas de frequência propícias a interferência. As frequências de corte e o projeto do filtro utilizado para realizar essas atenuações são descritos a seguir.

Frequências de corte. Em particular, este trabalho aplicou filtros *high-pass* e *low-pass* para atenuar ruídos nas faixas de frequência que são mais suscetíveis a ruídos tais como sinais DC e ruídos externos, tais como originadas de equipamentos eletrônicos (CHEVEIGNÉ; NELKEN, 2019). A filtragem *high-pass* permite que frequências acima de um limite de corte (*cuttoff*) fiquem inalteradas, mas atenuam frequências abaixo desse limite de corte. A abordagem CogEff define uma frequência de corte de 0,5Hz. Frequências abaixo desse limite de corte é reconhecido por conter comportamentos anormais causado por picos de atividade neural que são arbitrárias, e sinais com componentes DC. A filtragem *low-pass* realiza o contrário em relação a filtragem *high-pass*, pois não realiza alterações as frequências abaixo do limite de corte, mas rejeita frequências acima desse limite. O indicador CogEff utiliza uma filtragem *low-pass* de 45 Hz porque ela coincide com as frequências de dispositivos eletrônicos externos. A filtragem *high-pass* e *low-pass* foi baseada no filtro definido a seguir.

Tipo de filtro utilizado. Este trabalho utiliza um filtro do tipo Resposta ao Impulso Finita (FIR). O filtro FIR (PARKS; BURRUS, 1987) trata cada sinal como uma resposta de impulso finita. Isto porque, o filtro FIR soma uma faixa finita dos últimos sinais recebidos. A Equação 4.1, contida em PARKS; BURRUS (1987), apresenta a soma da convolução executada pelo filtro FIR.

$$y(n) = \sum_{m=n}^{n-N+1} h(n-m)x(m) \tag{4.1}$$

Onde:

- y(n) é o sinal de saída;

- x(m) é o sinal de entrada;

- h(n-m) é o impulso de resposta no instante n-m.

Especificamente, esse filtro foi projetado para utilizar o método *Hamming window* (PARKS; BURRUS, 1987; HARRIS, 1978), isto é, o filtro realiza as iterações sobre o sinal de modo janelado utilizando operações *Hamming*, de acordo com a Equação 4.2 (PARKS; BURRUS, 1987).

$$W(n) = 0.54 - 0.46 \cos \frac{2\pi n}{N-1} \tag{4.2}$$

Esta equação é aplicada nos filtros *high-pass* e *low-pass*. Especificamente, O filtro FIR irá iterar sobre o sinal, e aplicar a equação 4.2 dentro dos limites especificados nos limites de corte (ω_c). Para o filtro *high-pass* de 0,5Hz, uma função *Hamming window* |ω| é aplicada enquanto $\omega_c \leq$ 0,5Hz. O filtro *low-pass* com um limite de corte (ω_c) de 45Hz aplica a função *Hamming window* |ω| quando $\omega_c \leq$ 45Hz. A última etapa da filtragem trata-se de diminuir a quantidade de componentes relacionados aos movimentos oculares através da Análise de Componentes Independentes (ICA).

Análise de Componentes Independentes (ICA). Em relação a Figura 11, o último passo da Etapa 1 trata-se da análise Análise de Componentes Independentes (ICA) para a redução de componentes relacionados a movimentos oculares. Os sinais de EEG são geralmente compostos por mais de um componente, além de refletir a atividade neural, as quais são originadas de diferentes fontes, tais como, interferências de dispositivos externos, e movimentos oculares. A análise em componentes independentes é realizada pelo seguinte modelo generativo apresentado na Equação 4.3 (HYVÄRINEN; OJA, 2000). Onde x é um vetor de dimensão m, s é um vetor de dimensão n. s representa o sinal EEG com os componentes, e A é a matriz com dimensão mxn que o modelo deve estimar os componentes independentes. Após o algoritmo de ICA estimar A, é gerado uma matriz inversa W a partir de A (HYVÄRINEN; OJA, 2000).

$$x = As \tag{4.3}$$

Especificamente, este trabalho utiliza o método *fast* ICA para realizar a análise em componentes independentes (HYVARINEN, 1999). O método de *fast* ICA executa a análise de componentes independentes através da minimização de não-gaussianos. O *fast* ICA executa essa análise aproximando os sinais utilizando abordagem linear de pontos fixados (*linear point-fixed approach*). As Equações 4.4 e 4.5 representam as operações realizadas pelo *fast* ICA. Esta equação apresenta a estimativa dos componentes. Equação 4.4 subtrai os vetores *p* que foram computados anteriormente em $W_{p+1}^T C W_j W_j$ a partir de W_{p+1}. Por fim, Equação 4.5 normaliza W_{p+1} (HYVARINEN, 1999; HYVÄRINEN; OJA, 2000).

$$1.W_{p+1} = W_{p+1} - \sum_{j=1}^{p} W_{p+1}^{T} C W_j W_j \tag{4.4}$$

$$2.W_{p+1} = \frac{W_{p+1}}{W_{p+1}^{T} C W_{p+1}} \tag{4.5}$$

4.1.2 Etapa 2: Montagem do grafo e definições de propriedades

Esta etapa tem como objetivo montar o grafo responsável por representar a conectividade funcional durante as tarefas de compreensão de código. Este grafo é formado por nodos e arestas. Os nodos são a representação de cada canal de EEG utilizado para coletar as ondas cerebrais dos participantes. As arestas se constituem na conexão entre os nodos, e são estabelecidas caso exista uma sincronização entre dois canais distintos. Por último, esta etapa também define propriedades para estabelecer a relevância de cada aresta. Estas propriedades são um dos fatores que tornam a abordagem específica para compreensão de código. De acordo com a Figura 11, a Etapa 2 se inicia após os sinais de EEG forem filtrados na Etapa 1. A Etapa 2 consiste em três subprocessos: decompor o sinal de EEG, calcular o índice de conectividade entre os canais EEG, e atribuir o valor as propriedades das conexões. Estes subprocessos são definidos abaixo.

Decomposição do sinal EEG. A CogEff utiliza o algoritmo de transformada rápida de Fourier (*fast Fourier transform*) para transformar o sinal de EEG do domínio do temporal (*time-domain*) para um domínio discreto de frequência (*frequency-domain*). A transformada rápida de Fourier é definida na Equação 4.6 (COOLEY; TUKEY, 1965). Essa Equação recebe uma amostragem do sinal contínuo no domínio de tempo (x_m), e o transforma a um domínio de frequência ($\bar{X_k}$).

$$\bar{X_k} = \sum_{m=0}^{N-1} x_m W^{mk}, W = e^{\frac{-j2\Pi}{N}}, j = \sqrt{-1} \tag{4.6}$$

Após a transformada de Fourier, a CogEff extrai a fase do sinal. A fase de um sinal é relacionada ao ângulo da onda.

Cálculo do index de conectividade. Para estabelecer uma conexão entre dois nodos (isto é, canais de EEG), A abordagem CogEff calcula uma diferença de clusterização de fases baseado no tempo (*over-time phaze clustering difference*) entre duas séries temporais de sinais de EEG. *Over-time* significa que a conectividade é medida dentro de uma janela de dados do sinal. É uma técnica que calcula uma diferença *phaze-clustering* porque dentro desta janela, a técnica captura um cluster (ou agrupamento) de ângulos de fase entre os dois sinais. A diferença entre esses clusters de ângulos ajuda a quantificar a quão síncronos essas ondas estão. O índice de conectividade entre dois canais é apresentado na Equação 4.7 (COHEN, 2014).

$$Index = \left| \frac{e^{\Theta_{ch1_{t1}} - \Theta_{ch2_{t1}}} + ... + e^{\Theta_{ch1_{tn}} - \Theta_{ch2_{tn}}}}{tn} \right| \tag{4.7}$$

A Equação 4.7 calcula o índice de conectividade (ou sincronização) entre dois sinais originados de dois eletrodos EEG distintos. Essa conectividade ocorre em uma frequência de banda alfa. Este cálculo ocorre em uma janela de tempo, especificamente, a partir do tempo $t1$ até tn. O símbolo Θ indica o ângulo de fase do sinal em um tempo específico. Primeiro, a Equação 4.7 calcula a diferença entre fases do canal 1 ($ch1$) e o canal 2 ($ch2$) no tempo $t1$ ($\Theta_{ch1_{t1}} - \Theta_{ch2_{t1}}$). Em seguida, a Equação 4.7 calcula o número de Euler dessa diferença (e). A eularização de um número complexo é notação mais aceita para o comportamento oscilatório de sinais elétricos. Calcular o número de Euler da diferença entre os sinais de $ch1$ e $ch2$ é repetido até o último ponto de tempo na janela (n). Esses valores são então somados e então um média é obtida. Por fim, o valor absoluto dessa média resulta no índice de conectividade. Um valor maior que 0.3 e menor que 0.5 indica que os dois canais estão em sincronia, especificamente, na janela de tempo em que foi calculada. Um valor acima de 0.5 também indica sincronização oscilatória, porém é um valor considerado espúrio, i.e., gerado a partir de uma anormalidade do dado.

Grafo. O grafo utilizado na CogEff é definido por $G(v, e)$, onde G é um grafo representando um conjunto de vértices v, representando cada canal EEG. A localização dos canais no escalpo segue o padrão internacional 11-20. Os canais são restritos aos contidos no Emotiv Epoc+, pois foram os sensores utilizados no trabalho. Cada canal pode ser conectado através de arestas no conjunto e. As arestas e conectam um par de diferentes vértices no conjunto v. A CogEff utiliza um grafo não direcionado, isto é, para a CogEff o aspecto mais importante é a existência de conexões, e não a direção. Especificamente, porque inferir uma direção de conexão entre dois canais é possível, mas necessitaria de evidências empíricas para atribuir a sequência de iterações sobre qual canal influenciaria o outro. Por isso, a CogEff apenas constata pela presença de uma aresta que existe relação na interação entre duas regiões.

Atribuir valor às propriedades das conexões. A abordagem CogEff possibilita estabelecer pesos as arestas que conectam os canais de EEG. O objetivo deste passo é atribuir pesos para as conexões em relação ao quanto elas contribuem para compreensão de código. O peso é aplicado em caso de existência de uma aresta entre um par de nodos. É possível atribuir três tipos de propriedades, especificamente em relação aos aspectos de compreensão de código, hemisfério e lobo. Desenvolvedores podem atribuir pesos as conexões que estão em regiões cerebrais que ativam durante a tarefas de compreensão de código, conforme apresenta a literatura. Essas propriedades são descritas abaixo:

- **P1. Compreensão de código:** esta propriedade atribui um peso de 0 a 1 para conexões potencialmente relacionadas à atividade de compreensão de código (PEITEK et al., 2018a; SIEGMUND et al., 2017). Nessa propriedade, pode ser atribuído um valor próximo de um no caso de conexão estar fortemente associada a atividade de compreensão de código com base em evidências na literatura de engenharia de software (PEITEK et al.,

2018b; SIEGMUND et al., 2014);

- **P2. Hemisfério:** esta propriedade atribui um peso de 0 a 1 para quantificar a contribuição que o hemisfério cerebral onde se encontra a conexão contribui a compreensão de código (PALANIAPPAN, 2011). Evidências mostram que o hemisfério esquerdo é ativo durante atividades de compreensão de código (CRK; KLUTHE, 2016);

- **P2.1. Conexões Inter-Hemisféricas:** Alternativamente a P2, em caso a conexão abranger mais de um hemisfério, esta propriedade se refere a atribuir um peso de 0 a 1 para as arestas a qual conectam dois hemisférios (PALANIAPPAN, 2011);

- **P3. Lobo:** essa propriedade atribui um peso de 0 a 1 a conexões que estão dentro de um lobo cerebral específico (HUANG et al., 2019). Este peso tem o objetivo de atribuir a importância que lobos cerebrais tem na compreensão de código. Na literatura, o lobo frontal e central ativou durante tarefas de compreensão de código (CASTELHANO et al., 2019);

- **P3.1. Inter-lobo:** alternativamente a P3, em caso de a conexão abranger a interação de dois lobos diferentes esta propriedade passa a atribuir o peso de relevância entre diferentes lobos. Especificamente, é a interação em que arestas conectam um par de canais localizados em partes distintas do lobo cerebral.

O peso total de uma conexão é definido como uma média ponderada do peso de cada propriedade. Por exemplo, suponha uma conexão entre o canal AF3 e F3. Assumindo que a conexão entre esses canais é correlacionada às tarefas de compreensão de código, e por isso um peso de 0.8 é atribuído a essa conexão (P1 = 0.8). Esta conexão também está situada no hemisfério esquerdo, na qual a literatura alega que é extensivamente ativada durante compreensão de código, deste modo podemos atribuir 1 para a propriedade P2 (P2 = 1). Por fim, esta conexão se encontra no lobo frontal, onde também a literatura assume que existe frequente atividade durante tarefas de compreensão, e então, pode-se atribuir, por exemplo, 0.7 a propriedade P3 (P3 = 0.7). Desse modo, a média ponderada para as propriedades dessa aresta é calculada na seguinte maneira $\frac{P1+P2(or P2.1)+P3(or P3.1)}{3}$. Especificamente, para os pesos atribuídos nesse exemplo é obtido um peso de 0.83 para a conexão AF3 - F3, isto é, $\frac{(0.8+1+0.7)}{3} = 0.833$.

Exemplos para motivação das propriedades. Estudos em engenharia de software mostraram que áreas cerebrais específicas são ativadas durante tarefas de compreensão de código. Assim que a literatura avança nas evidências na relação entre compreensão de código e ativação de áreas cerebrais. As propriedades podem ser atribuídas para customizar a relevância dessas conexões em relação as descobertas da literatura, com a finalidade de ocorrer ajustes. Outro aspecto importante é que pesos ajudam a evitar ambiguidade nos resultados, no sentido de que conexões que são de maior interesse e ligam regiões relacionadas a compreensão de código possuam mais importância a outras conexões que podem ser atribuídas a outras atividades cognitivas, tais como, movimentos oculares. A Figura 12 apresenta alguns exemplos entre

pares de canais com as propriedades definidas anteriormente. Figura 12 apresenta arestas que conectam canais EEG de três desenvolvedores (Dev. 1, Dev. 2 e Dev. 3). Pode ser assumido que a CogEff poderia apresentar o mesmo resultado para esses três cenários de conexão, sem a existência das propriedades. Isto implicaria em um resultado que consideraria igualmente áreas não relacionadas a compreensão de código com as áreas relacionada de maneira equivalente. Especificamente, se todos os pesos fossem o mesmo no exemplo da Figura 12, CogEff apresentaria o mesmo resultado para as conexões extraídas do Dev. 3, Dev. 2, e Dev. 1. Contudo, a Figura 12 apresenta que diferentes pesos levam a interpretação de que conexões do Dev. 1 tem mais importância. Além disso, horizontalmente a Figura 12 apresenta grafos conectados que representam a conexão de canais de EEG de três diferentes desenvolvedores. Verticalmente, a Figura 12 apresenta pesos atribuídos em relação as propriedades da CogEff.

Em relação a notação dos grafos contido na Figura 12, a Table 24 descreve as notações utilizadas para cada cenário.

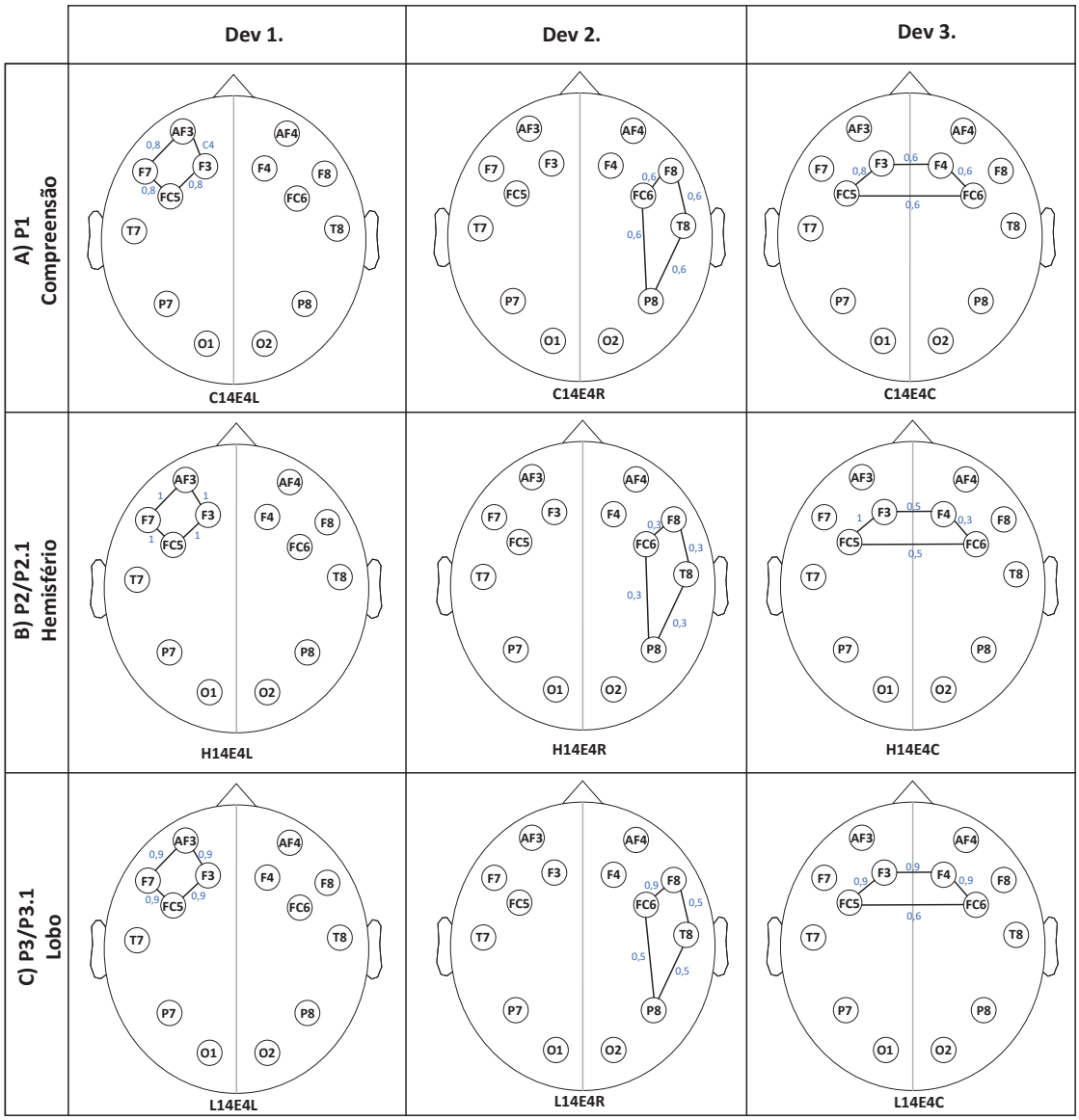
Tabela 24: Legenda da notação contida na Figura 12.

Símbolo	**Descrição**
C_n	Um grafo com pesos atribuídos a compreensão com n nodos
H_n	Um grafo com pesos atribuídos ao hemisfério cerebral com n nodos
L_n	Um grafo com pesos atribuídos ao lobo cerebral com n nodos
E_n	Quantidade de arestas presente no grafo
L	Conexões concentradas no lado esquerdo
R	Conexões concentradas no lado direito
C	Conexões concentradas no região central

Os pesos definidos na Figura 12 são baseados no seguintes critérios:

- **P1. Compreensão de código:** Figura 12.A apresentam os pesos em relação ao aspecto de compreensão de código as conexões existentes. De acordo com as evidências na engenharia de software (SIEGMUND, 2016; PEITEK et al., 2018a; FIGL, 2017), as regiões frontais e centrais são frequentemente ativadas durante tarefas de compreensão de código. Por isso, para cada conexão foi atribuída um peso de 0.8 nas conexões do Dev. 1. Dev. 2 gerou um grafo com 4 conexões no lado direto do cérebro, e não existem evidencias que esta região está envolvida na compreensão de código, então são atribuídos pesos de 0.6 para as arestas. O Dev. 3 contém um grafo de conexões em algumas regiões que são relacionadas a compreensão de código (canais FC5 e F3), e outras áreas que não são relacionados (canais FC6 e F4). Neste caso, a conexão entre canais que envolvem áreas relacionados a compreensão de código (F3 e FC5) possuem maior pesos em relação as conexões envolvendo áreas sem relação. Neste critério, o grafo do Dev. 1 obteve um peso maior que o grafo do Dev. 2, e Dev. 3. Ainda temos a informação de que o grafo do Dev. 3 tem peso maior que o grafo do Dev. 2. Este peso maior é consequência do envolvimento de conexões entre canais F3 e FC6 no grafo do Dev. 3. Os pesos também possibilitam

Figura 12: Exemplos de conexões com pesos de acordo com as propriedades definidas.



Fonte: Elaborado pelo autor.

balancear o peso de conexões que conectam regiões relacionadas a compreensão de código, a regiões que não relacionadas, desta forma pode-se atribuir um decréscimo de peso nessas regiões, por exemplo, o ocorre na conexão FC6 e FC5;

- **P2/P2.1. Hemisférico/Inter-hemisférico:** Figura 12.B apresenta os pesos do aspecto hemisférico/Inter-hemisférico. De acordo com CASTELHANO et al. (2019); SIEGMUND et al. (2014); CRK; KLUTHE (2016), as áreas cerebrais do hemisfério esquerdo são ativadas durante a compreensão de código. Baseado nisso, os pesos neste exemplo serão mais altos naquelas conexões que ligarem regiões concentradas no hemisfério esquerdo. As conexões do grafo do Dev. 1. possuem peso 1. No grafo do Dev. 2, as conexões no hemisfério direito foram atribuídas o peso 0.3 por causa da pouca evidência de que esse hemisfério se envolve durante tarefas de compreensão de código. No

grafo do Dev. 3 existem conexões que se enquadram na regra inter-hemisférica (P2.1), em particular, as conexões entre os pares F3 e F4, e FC5 e FC6. Como essas conexões pertencem a hemisférios diferentes elas recebem um peso menor aos atribuídos nos hemisférios esquerdo e maior do que conexões no hemisfério direito. No geral, o grafo do Dev. 1 alcançou peso máximo porque todas as conexões estão concentradas no hemisfério esquerdo. O grafo do Dev. 2 recebeu um peso menor porque todas as conexões se concentraram no hemisfério direito. O grafo do Dev.3 possui mais peso que o Dev. 2 porque existem conexões interagindo com o hemisfério esquerdo, e pesos menores que as conexões do grafo do Dev. 1 por causa da presença das conexões no lado direto.

- **P3/P3.1. Lobo/Inter-Lobo:** Figura 12.C apresenta os pesos no aspecto lobo/inter-lobo atribuídas em relação as arestas. De acordo com as evidências em (CASTELHANO et al., 2019; PEITEK et al., 2018a,b; SIEGMUND, 2016) os lobos frontal, central e parietal são geralmente ativados durante a compreensão de código. Por isso, conexões envolvendo áreas temporais e occipitais receberam pesos menores em relação a aquelas conexões que envolvem lobos frontal, central e parietal. De acordo com esse critério, Dev. 1 tem mais peso que grafos do Dev. 2 e Dev. 3. O grafo do Dev. 2 possui pesos menores pois envolvem os lobos em que se encontram menos evidências relacionadas a compreensão de código.

4.1.3 Etapa 3: Densidade do grafo

Em relação a Figura 11, esta etapa tem o objetivo de calcular a densidade do grafo formado na Etapa 2. Depois de aplicar e definir os pesos as arestas do grafo, a abordagem CogEff obterá a densidade. Especificamente, é utilizada a densidade ponderada do grafo (LIU; WONG; CHUA, 2009; UNO, 2007). Na teoria dos grafos, a densidade de um grafo é importante porque quantifica o quão conectado está, e indica também como um determinado grafo difere de outros. Assim, no caso deste trabalho, mostraria a importância dos grafos com mais conexões na área relacionada à compreensão de código. A densidade de um grafo também é importante para distinguir como um grafo é diferente de outro com base em aspectos como a quantidade de conexões e seu peso. A Equação 4.8 calcula a densidade de um grafo com pesos atribuídos as arestas (LIU; WONG; CHUA, 2009).

$$\textit{Densidade do Grafo} = \frac{\sum_{v \in V, u \in V, v \neq u} Weight(u, v)}{(|V|.(|V-1|))/2} \tag{4.8}$$

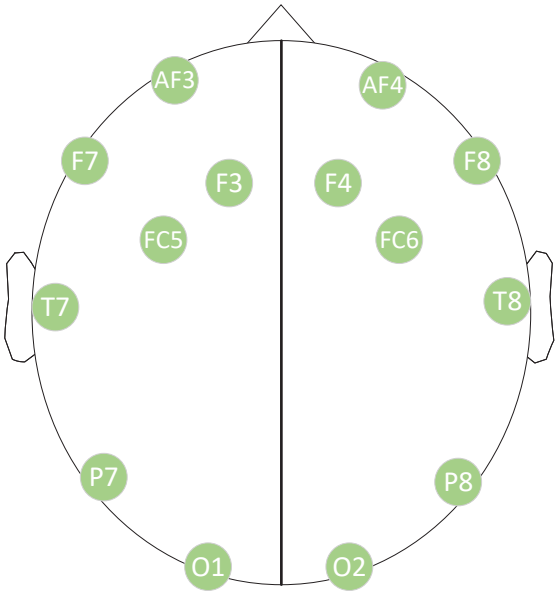
A Equação 4.8 realiza uma soma ($\sum$) dos pesos das conexões entre os nodos u e v, e divide pela quantidade possível de conexões entre os vértices $(|V|.(|V-1|))/2$ em um grafo não direcionado. O valor de saída varia entre 0 e 1, onde 0 valores próximos a 0 significam um grafo fracamente conectado e de baixa densidade, e valores próximos a 1 significam um grafo

com grande quantidade de conexões e alta densidade.

4.2 Instrumentação

A *CogEff* mensura a carga cognitiva obtida de desenvolvedores baseada nas leituras obtidas a partir de um eletroencefalograma. O eletroencefalograma captura sinais elétricos gerados pela atividade neural de desenvolvedores de software, geralmente sinais elétricos que variam entre 5–200 microvolts. O EEG captura ondas cerebrais, os quais possuem um sinal fraco, pois os eletrodos são posicionados no escalpo, e dessa forma, possui uma barreira com a superfície cerebral. Por ser posicionada no escalpo, trata-se de um instrumento não invasivo, diferentemente de instrumentos de eletroencefalograma invasivos, as quais necessitariam ter os eletrodos implantados acima do tecido cerebral, mais conhecido como eletroencefalografia intracraniano (iEEG). São motivos pela adoção do EEG neste trabalho: a facilidade do manuseio, pois trata-se de um dispositivo portátil, a existente correlação na literatura com processos cognitivos (COHEN, 2017; CRK; KLUTHE, 2016), e alta precisão temporal, refletindo estímulos relacionadas a tarefas com instantaneidade, diferentemente de dispositivos de baixa precisão temporal, como o fNIRS, o qual reflete estímulos relacionados a atividades com média de 10 segundos (PEITEK et al., 2018a) após os eventos. Esse atraso ocorre porque ele reflete o processo de fluxo sanguíneo cerebral. Neste trabalho, o aparelho adotado para capturar as ondas cerebrais do usuário é o Emotiv EPOC+ (Emotiv Systems, 2021), um eletroencefalograma portátil composto por 14 canais (eletrodos) posicionados de acordo com o padrão internacional 10-20, o qual é ilustrado na Figura 13, e realiza uma leitura de 256Hz das ondas cerebrais, isto é, gera leituras de 256 linhas por segundo.

Figura 13: Canais do Emotiv EPOC de acordo com o padrão internacional 10-20.



Fonte: Elaborado pelo autor.

Banda de frequência. As ondas cerebrais obtidas do eletroencefalograma podem ser decompostas em várias frequências, tais como, alfa (α) [8-13Hz], beta (β) [13-30HZ], theta (θ) [8-13Hz], delta (δ) [1-4Hz], e gama (γ) [30-70Hz]. A abordagem CogEff utiliza todas as frequências de banda, isto é, a banda alfa (α) [8-13Hz], Beta (β) [13-30HZ], Theta (θ) [8-13Hz], Delta (δ) [1-4Hz], e Gama (γ) [30-70Hz] para mensurar a carga cognitiva dos desenvolvedores de software em tarefas de compreensão de código. Todas as bandas são utilizadas por causa da relação dessas bandas com os vários processos cognitivos importantes tais como, memória de trabalho (*working memory*),(CRK; KLUTHE; STEFIK, 2015; CRK; KLUTHE, 2016), estado de alerta e atenção, e raciocínio lógico (KOSTI et al., 2018).

Canais. Alguns trabalhos utilizando o eletroencefalograma (CRK; KLUTHE, 2016)(LEE et al., 2017) restringiram a medição das ondas cerebrais selecionando alguns eletrodos na parte frontal do cérebro, argumentando que a literatura de ressonância magnética na engenharia de software, apontou a ativação dessas áreas específicas (áreas de *Broadmann* 44 e 47) durante tarefas de compreensão de código (SIEGMUND et al., 2017; PEITEK et al., 2018a). Porém, de acordo com a própria literatura, a ativação de áreas durante a compreensão de programas não se restringe a região frontal. A região central também está envolvida, tais como a área de *Broadmann* 40, 21, 6, e 7. Além do EEG ser um equipamento de baixa precisão espacial (não possui a mesma precisão do fMRI para indicar a fonte da atividade espacial), o equipamento utilizado é limitado a 14 canais. De acordo com as evidências da literatura, praticamente todas as regiões cerebrais têm envolvimento durante a tarefa de compreensão de código, mas, com a maioria das áreas ativas no lado esquerdo (SIEGMUND et al., 2017; PEITEK et al., 2018a). Desta forma, a abordagem CogEff, tem foco nos canais do lado esquerdo, em particular os canais AF3, F3, F7, FC5, T7, P7 e O1.

4.3 Tarefas

A abordagem CogEff foi concebida para mensurar a carga cognitiva em tarefas de compreensão de código no formato geralmente empregado nos experimentos em engenharia de software. A definição das tarefas de compreensão de código empregadas em estudos experimentais consiste na tarefa do desenvolvedor em ler, interpretar um trecho de código, e deduzir o resultado. A Figura 14 ilustra uma tarefa de compreensão de código tipicamente encontrada em estudos experimentais para avaliar abordagens para mensurar carga cognitiva. Essas tarefas consistem em código fonte na linguagem Java, em uma média de 10 a 20 linhas. Tarefas são apresentadas aos desenvolvedores, e eles devem escolher uma das cinco opções disponíveis. A partir disso, existem três possibilidades para as respostas: errada, correta, ou sem resposta em caso de desenvolvedores não responderem à questão.

Eventos no eletroencefalograma. As tarefas são precedidas por momentos de relaxamento, onde o usuário é orientado a permanecer ocioso. Após o relaxamento, o usuário inicia a tarefa de compreensão, e responde à questão. Esses eventos (relaxamento, tarefa e resposta) são iden-

Figura 14: Exemplo de tarefa de compreensão de código geralmente empregada nos estudos experimentais.

```
1.  public int count() {
2.     String sentence1 = "I Went To The Super Market";
3.     String sentence2 = "If i miss the buss i will arrive late to the job";
4.     int size = 0;
5.
6.     if (sentence1.length() >= sentence2.length()) {
7.         for (int i=0; i<sentence1.length(); i++) {
8.             size++;
9.         }
10.    }else {
11.        for (int i=0; i<sentence2.length(); i++) {
12.            size++;
13.        }
14.    }
15. return size;
16.
17. }
```

A) 50 B) 49 C) 26 D) 26 E) 38

Fonte: Elaborado pelo autor.

tificados nos dados do eletroencefalograma através de marcadores específicos, os quais são inseridos no EEG. Nesse trabalho, a duração das tarefas tem um limite de 60 segundos.

Medidas tradicionais da carga cognitiva. Para avaliar a correlação da CogEff com compreensão de código, foram coletadas medidas que são utilizados em estudos tradicionais de carga cognitiva, tais como, as respostas das tarefas de compreensão de código, e o tempo decorrido para concluir a tarefa de compreensão (CRK; KLUTHE, 2016). Estes dados, são considerados em estudos de carga cognitiva para avaliar processos mentais, no caso deste estudo, compreensão de código.

4.4 Escala da carga cognitiva da CogEff

Esta Seção define a escala do valor da abordagem CogEff. Esta escala é definida em relação aos conceitos sobre a teoria da mensuração apresentada em WOHLIN et al. (2012). O resultado da abordagem CogEff é um valor que pertence a um intervalo de 0 a 1 [0,1], onde, especificamente, este intervalo pode ser interpretado em 5 escalas:

- 0,0 a 0,2: Muito baixo;
- 0,2 a 0,4: Baixo;
- 0,4 a 0,6: Moderado;
- 0,6 a 0,8: Alto;
- 0,8 a 1,0: Muito Alto.

O valor da CogEff é subjetivo, pois pode sofrer alterações por depender tanto do objeto sendo medido quanto de fatores humanos de julgamento. Em particular, o esforço cognitivo

aplicado do usuário pode mudar, caso seja medido novamente pela mesma pessoa, mesmo realizando a mesma tarefa.

4.5 Implementação

Esta Seção apresenta os algoritmos que fazem parte da abordagem CogEff em relação a definição contida na Seção 4.1. A Seção 4.5.1 apresenta o algoritmo para filtragem de sinais EEG. A Seção 4.5.2 apresenta o algoritmo para montar o grafo utilizado para calcular a influência das arestas no valor final. A Seção 4.5.3 apresenta o algoritmo para calcular a densidade do grafo.

4.5.1 Filtragem EEG

O algoritmo 1 detalha os procedimentos para filtrar o dado EEG. Este é um passo importante para reduzir os ruídos contidos no sinal de EEG. A linha 1 define os parâmetros da Análise de Componentes Independentes (ICA). É definida nessa linha que o método ICA será o *fastICA*. Há no máximo 13 componentes nos canais EEG, e foram definidas 10.000 interações para realizar o ICA. Linhas 2 e 3 aplicam os filtros *low-* e *high-pass* nos dados EEG. Especificamente, os filtros foram projetados para tratar as frequências de corte abaixo de 0,5 Hz e acima de 45Hz.

Algoritmo 1: Algoritmo para filtragem do sinal EEG

Entrada: A matriz CT_{EEG} de dimensão $m \times n$ com dados EEG dos desenvolvedores;
Saída : A matriz CT_{EEG} de dimensão $m \times n$ com dados EEG filtrados dos desenvolvedores;

1 $ica \leftarrow$ *definir parâmetros:* $n\_components = 13, random\_state = 0, method =' fastica', max\_iter = 10000$;
2 *Aplicar filtro low-pass em* CT_{EEG};
3 *Aplicar filtro high-pass em* CT_{EEG};
4 *Executar filtro FIR em* CT_{EEG} *com a opção* fir_params *nas frequências* $<$ *0.5Hz e* $>$ *45 Hz*;
5 $eye\_bad\_artefacts \leftarrow$ *encontrar posição de artefatos oculares em* CT_{EEG} *baseados nos dados do canal F7*;
6 *Remover artefatos oculares* CT_{EEG} *baseado nas posições definidas em* $eye\_bad\_artefacts$;

Fonte: Elaborado pelo autor.

Linha 4 define que o tipo de filtro a ser utilizado é o *Finite Impulse Response* (FIR). Linha 5 utiliza o algoritmo ICA para encontrar artefatos oculares nos dados EEG, em cada um dos canais. Linha 6 remove os artefatos encontrados no sinal com base nas posições encontradas na linha 5.

4.5.2 Montagem do grafo e atribuição de pesos

O algoritmo 2 é responsável por montar o grafo da CogEff. A linha 1 define a taxa de amostragem das leituras das ondas EEG utilizada pela CogEff. Esta linha é atribuída a taxa de amostragem utilizada pelo equipamento que coleta ondas cerebrais, o qual tem uma taxa de amostragem de 256Hz. A linha 2 o tamanho da janela de dados em segundos. A linha 3 traduz o tamanho em segundos para o tamanho de linhas que essa janela alcança. A linha 4 contém a lista de canais selecionados, a qual a CogEff utiliza para montar o grafo, que praticamente compreende em todos os sensores disponibilizados pelo aparelho utilizado.

A linha 5 define um *array* a qual é utilizado para armazenar os resultados de cada janela. A linha 7 cria a instancia de um grafo vazio com 14 nodos, respectivos aos canais EEG que coletaram dados. Na linha 8 um laço *while* itera enquanto houve dados EEG a serem processados. Linha 9 calcula o tamanho da atual janela. Quando uma janela de dados de EEG termina de ser processada, a primeira posição da janela (*f_idx_task*) recebe o valor da última posição desta janela (*l_idx_task*).

Desta forma, se define que, o início da próxima janela, é após o término da janela atual. A última posição é obtida através do acréscimo do valor definido na variável *janela*. Na linha 10, é verificado se o tamanho da janela atual é menor que valor restante a ser processado dos dados. Se verdadeiro, isso indica que se trata da última janela a ser processada para o dado de entrada.

A linha 11 verifica se a janela é maior que aproximadamente 10% da taxa de amostragem, isto é, mais ou menos equivalente a 20 linhas de leitura, pois janelas abaixo desse limite não fornecem dados suficientes para convergir no cálculo da potência de banda (*bandpower*). Entre as linhas 13 e 14, o algoritmo executa o cálculo de *bandpower* de cada canal do EEG.

Entre as linhas 15 a 27 o algoritmo estabelece a conectividade entre os nodos (canais EEG). Linhas 15 e 16 ocorrem os laços de interações que iteram sobre a lista de canais especificadas no *array* chn_lst. A linha 17 verifica se os nodos são diferentes, pois nesse grafo, os nodos não se conectam consigo mesmos. A linha 19 calcula a transformada de Fourier do sinal utilizando o método *fast Fourier transform (fft)*. A linha 20 extrai a fase da frequência calculada na linha 19.

Linhas 21 a 23 calculam os mesmos procedimentos para o outro canal. A linha 24 calcula a diferença de fases de cada ponto temporal do sinal. A linha 25 transforma a diferença de fases em uma representação de números complexos. A linha 26 calcula o número de Euler dos números complexos obtidos na linha 25. A linha 27 calcula o índice de conectividade entre os dois canais. A linha 30 estabelece a conexão se o índice de conectividade resultar em um valor na faixa de 0.3 a 0.5 (condição da linha 28). Os pesos é atribuído a aresta de acordo com os valores definidos a cada propriedade na linha 29.

Fonte: Elaborado pelo autor.

Algoritmo 2: Montagem de grafo e estabelecimento de conexões

Entrada: Sinais EEG filtrados;
Saída : double 0..1

```
1  taxa_amostragem ← 256;
2  segundos ← 10;
3  janela ← taxa_amostragem*segundos;
4  chn_lst ← ["AF3", "F7", "F3", "FC5", "T7", "AF4", "F8", "F4", "FC6", "T8", "P7", "O1",
     "P8", "O2"];
5  resultado_arr;
6  grafo_arr;
7  G = mountGraph ();
8  while true do
9      tam_window ← l_idx_task - f_idx_task;
10     if tam_window < window then
11         if tam_window > 20 then
12             for i = 0; i < chn_lst.size; i++ do
13                 resultado_arr[hn_lst[i]] ← bandpower (hn_lst[i], EEGData(f_idx_task,
                     l_idx_task,'alpha',sample_rate))
14             end
15             for i = 0; i < chn_lst.size; i++ do
16                 for j = 0; j < chn_lst.size; j++ do
17                     if i ! = j then
18                         signal_ch1 ← EEG[chn_lst[i]];
19                         spectro_1 ← fft (signal_ch1);
20                         fase_1 ← getPhase (spectrum_1);
21                         signal_ch2 ← EEG[chn_lst[j]];
22                         spectro_2 ← fft (signal_ch2);
23                         fase_2 ← getPhase (spectrum_2);
24                         diff_fases ← phase_1 - phase_2;
25                         complex ← 1i * diff_fases;
26                         euler_complex ← e^complex;
27                         index_con ← ∑euler_complex / timePoints;
28                         if index_con > 0.3 and index_con < 0.5 then
29                             weight ← properties;
30                             G.connect(chn_lst[i], chn_lst[j],weight)
31                 end
32             end
33         break;
34     break;
35 end
```

4.5.3 Densidade do grafo

O algoritmo 3 apresenta como é calculado um resumo do valor de todas as conexões do grafo obtido na Seção 4.5.2. A entrada desse algoritmo é o grafo obtido no algoritmo anterior, na Seção 4.5.2. A saída é o resultado da CogEff, um valor no intervalo entre zero e um. Para esse cálculo, o algoritmo utiliza uma equação de densidade do grafo baseado em pesos. Para isso, a linha 1 realiza a soma dos pesos de cada aresta. Em seguida, a linha 2 calcula a quantidade possível de conexões nesse grafo. A linha 3 do algoritmo 3 calcula o valor final da densidade de grafo baseado em pesos.

Fonte: Elaborado pelo autor.

Algoritmo 3: Densidade do grafo

Entrada: Um grafo G não dirigido composto por um conjunto de nodos V, e um grupo de arestas a;

Saída : double 0..1

1 $soma\_pesos\_arestas \leftarrow \sum_{v \in V, u \in V, v \neq u} Peso(u,v)$;

2 $quantidade\_conexoes \leftarrow (|V|.(|V-1|))/2$;

3 $densidade\_grafo \leftarrow \frac{soma\_pesos\_arestas}{quantidade\_conexoes}$

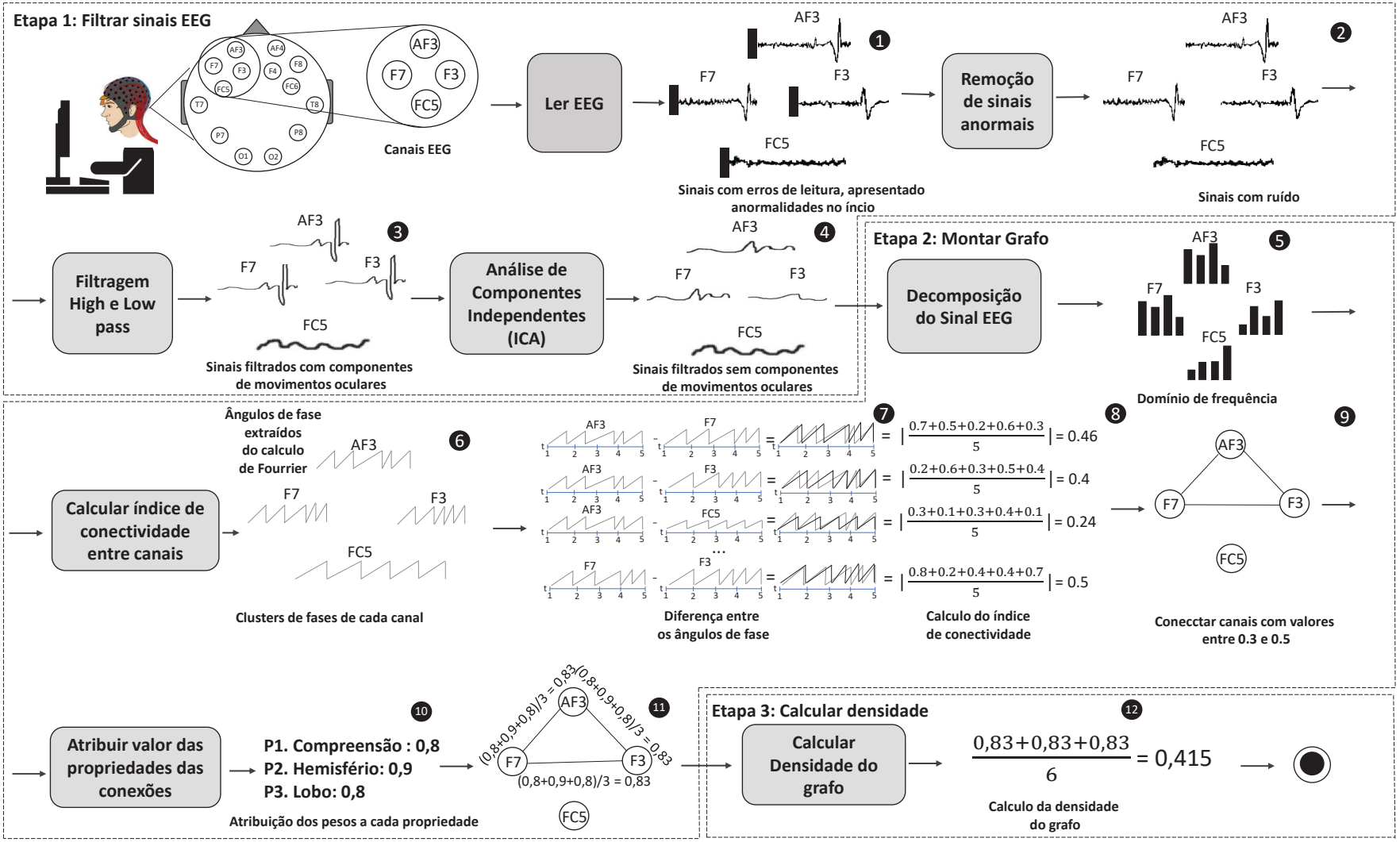
4.6 Demonstração da Execução da Abordagem CogEff

A Figura 15 representa uma demonstração da abordagem CogEff em relação ao processo apresentado neste Capítulo. O processo da abordagem CogEff para calcular a carga cognitiva em tarefas de compreensão de código compreende em três etapas.

Na *Etapa 1: Filtragem dos dados EEG*, a abordagem CogEff faz a leitura dos dados EEG com o objetivo de isolar um sinal limpo e apropriado para mensurar a carga cognitiva. Os sinais de EEG podem ser capturados com alguns erros que prática formam uma faixa escura inutilizável. Para isso, a abordagem realiza a **remoção de sinais anormais**. O resultado é ainda um sinal que contém ruídos e necessita passar por mais um processo de filtragem. Para isso, a abordagem aplica a **Filtragem *High* e *Low-pass***, a qual é responsável por aplicar filtros para remover sinais baseados nas frequências de corte. O resultado desse processo é um sinal limpo, porém ainda com componentes relacionados a movimentos oculares, o qual é possível notar através da grande curva que os sinais capturados dos canais AF3, F7 e FC5 possuem. Para remover esses componentes, a abordagem aplica a **Análise de Componentes Independentes (ICA)**, e remove tais componentes dos sinais EEG. Consequentemente, os resultados são sinais filtrados e sem componentes de movimentos oculares, ilustrados no final da Etapa 1 na Figura 15.

Na *Etapa 2: Montar grafo*, na Figura 15, a abordagem CogEff monta o grafo com base nos sinais filtrados. O grafo representa a conectividade funcional entre canais. Para isso, o primeiro passo da CogEff na Etapa 2 é **realizar a decomposição do sinal EEG**. O resultado deste passo é o sinal de cada um dos canais (AF3, F7, F3 e FC5) transformado para o *domínio de frequência* (Figura 15.5). O próximo passo consiste em **calcular o índice de conectividade entre canais**. Para calcular esse índice, primeiramente é obtido a fase das ondas de cada canal, que são as inclinações das oscilações das ondas (Figura 15.6). Após isso, é realizado a diferença entre os ângulos de fases entre cada par de canais de EEG (Figura 15.7). Em seguida, a fórmula do índice de conectividade *over-time* é aplicada, realizando o somatório, e dividido pelos pontos de tempo (Figura 15.8).

Figura 15: Exemplo em execução da abordagem CogEff.



Fonte: Elaborado pelo autor.

Em seguida, as conexões são estabelecidas conforme as regras do índice de conectividade: conectar canais com índices entre 0,3 e 0,5. Abaixo de 0,3 indica que não há conectividade entre a atividade de dois canais, acima de 0,5 indica que provavelmente a atividade origina da mesma fonte e não pela independência entre os dois canais. Com base nos resultados, os canais AF3, F7 e F3 estão conectados e apenas o canal FC5 não está conectado aos demais (Figura 15.9). Com as conexões estabelecidas o próximo passo é **Atribuir valor das propriedades as conexões**. Neste exemplo, é definido o valor da propriedade de compreensão em 0,8, hemisfério em 0,9, e lobo em 0,8 (Figura 15.10). Em seguida, a média ponderada entre esses valores definidos são calculadas (Figura 15.11).

Na *Etapa 3: Calcular densidade*, é calculado a densidade do grafo gerado na etapa anterior. Para isso, na Figura 15 é realizado o único passo dessa etapa, que é **calcular densidade do grafo**. Para essa densidade, são somados os pesos das conexões existentes seguido pela divisão da quantidade de conexões que podem existir em um grafo não direcionado, isto é, (|V.(V-1)|)/2 = (|4.(4-1)|)/2 = 12/2 = 6. Por fim, o resultado da abordagem CogEff, para esse instante específico em alguma tarefa de compreensão de código é de 0,415 (Figura 15.12).

4.7 Análise comparativa das abordagens correlacionadas

Este capítulo descreve as principais abordagens utilizadas para medir a carga cognitiva dos desenvolvedores na área de Engenharia de Software. As abordagens consistem no método utilizado para medir a carga cognitiva, e são compostos por métodos filtragem utilizados, e indicadores psicofisiológicas, tais como o ERD, e ASR. A seguir esse trabalho descreve as principais abordagens tradicionais utilizadas na literatura.

***Band Power* ou Potência de sinal:** neste trabalho, o *band power*, se refere a abordagem de medição da carga cognitiva utilizada por (KLIMESCH, 1999) para quantificar as bandas alfa e theta para mensurar a carga cognitiva (CRK; KLUTHE, 2016). Esta abordagem é utilizada em grande parte dos trabalhos que adotaram o EEG na área de engenharia de software (CRK; KLUTHE, 2016; CRK; KLUTHE; STEFIK, 2015) e foi também motivação para se usar e aplicar ondas frequências de banda além da banda alfa e theta (LEE et al., 2017). Em particular a potência de sinal consiste em mensurar a contribuição que faixas de frequência específicas tem em relação a todo sinal do EEG. A abordagem consiste em dois principais passos: (1) adotar um filtro *high* e *low - pass* e após isso, computar a potência de sinal da frequência de banda desejada, as quais foram capturadas durante as tarefas de compreensão de código. As quatro principais frequências de banda são alfa [8-12 Hz], beta [12-30 Hz], delta [0.5-4 Hz], e theta [4-8 Hz]. Utilizar essa abordagem pode compreender na adoção de diferentes frequências de bandas, as quais são relacionadas memória de trabalho (*working memory*), compreensão, e nível de dificuldade (KLIMESCH, 1999).

- **Delta [0.5-4 Hz]:** a presença de altas frequências e baixas amplitudes na banda delta indica estado de sono profundo. A presença de frequências rápidas e altas amplitudes

significa o estado de alerta dos desenvolvedores de software;

- **Theta [4-8 Hz]:** a atividade com menor frequência da banda theta indica a execução de tarefas lógicas e o uso de memória de trabalho e carga de trabalho;
- **Alfa [8-12 Hz]:** a frequência da atividade da banda alfa está relacionada a alta carga de trabalho durante a execução de tarefas lógica;
- **Beta [12-30 Hz]:** a atividade dessa banda indica foco e alerta. A presença de baixa amplitude e maior rapidez pode indicar esforço cognitivo na resolução de tarefas lógicas;

***Event-Related Desyncrhonization* (ERD):** A dessincronização relacionada a eventos é a abordagem para mensurar a carga cognitiva adotada por (CRK; KLUTHE, 2016; CRK; KLUTHE; STEFIK, 2015). A abordagem de calcular o ERD consiste em dois principais passos: (1) filtrar o sinal com o *high* e *low* pass (2) calcular a potência alfa do EEG durante o período de descanso, e também para o período gasto durante a tarefa de compreensão de código (3) calcular o ERD com base nesses dois valores alfa. Na ausência de atividade dos neural, a oscilação das bandas de frequência está síncrona, isto é, ambas em alta amplitude. Isto significa que, a ausência de atividade consiste no estado de relaxamento do usuário. Em contraste, quando aumenta a atividade neural as bandas de frequência ficam dessincronizadas, isto é, em baixa amplitude. A dessincronização relacionada a eventos quantifica a dessincronização que ocorre durante as tarefas de compreensão, em relação às ondas cerebrais sincronizadas, em estado de relaxamento.

***Asymmetry Ratio* ou taxa de assimetria (ASR):** a ASR consiste na abordagem para mensurar a carga cognitiva adotada por DURAISINGAM; PALANIAPPAN; ANDREWS (2017), a qual consiste em dois principais passos: (1) aplicação de um filtro *bandpass* do tipo IIR elíptico; e (2) cálculo da taxa de assimetria entre canais de EEG obtida durante as tarefas de compreensão de código. A taxa de assimetria mede a contribuição entre o hemisfério esquerdo e direito. O cálculo deve considerar um canal de EEG do hemisfério esquerdo em relação a outro canal EEG do hemisfério direito. O cálculo pressupõe que o hemisfério esquerdo do cérebro é usado para tarefas cognitivas lógicas e o hemisfério direito é usado durante tarefas visuais (DURAISINGAM; PALANIAPPAN; ANDREWS, 2017). Portanto, uma tarefa que exige tarefas lógicas dos desenvolvedores de software deve inferir um valor positivo, e tarefas visuais devem resultar em uma valores negativos.

***Phaze-Locking Value* (PLV):** O PLV refere-se a abordagem para medir a carga cognitiva utilizada por (KOSTI et al., 2018) estabelecendo a conectividade entre canais. Esta abordagem consiste em dois passos principais (1) aplicação de filtro *bandpass*; (2) estabelecer o PLV entre os canais de EEG. O PLV não é uma abordagem *over-time*, mas sim, *over-trial*, isto é, para ser estimado depende de coletar múltiplas amostragens das leituras de EEG da mesma tarefa de compreensão. O PLV é especificamente uma abordagem que estabelece conexões baseadas na coerência de fase. Para se obter a fase, é necessário realizar a transformada de Fourier da leitura

de EEG para o domínio de frequência. Desta forma, é possível obter um número imaginário no qual se pode derivar a fase e a magnitude. Se dois canais de EEG tiverem a fase síncrona, há uma relação entre esses dois canais. Em outras palavras, os sinais são síncronos se apresentarem um comportamento semelhante nos valores de fase.

Sinal BOLD: A abordagem BOLD neste trabalho se refere a abordagem adotada por (SIEGMUND et al., 2017; PEITEK et al., 2019; SIEGMUND et al., 2014) para medir a carga cognitiva através do fMRI. É a única abordagem escolhida para essa análise comparativa que não é obtida através do dispositivo de EEG. Essa abordagem consiste em dois passos: (1) preparação dos dados coletados com correções de movimentos, e filtragem dos dados na banda *high-pass*; (2) análise dos dados BOLD em imagem dos locais do cérebro ativados durante tarefas de compreensão. Em particular, através dos dados *blood oxygenation level dependent* (BOLD) é possível estimar as áreas ativas do cérebro em uma tarefa cognitiva pelo fluxo sanguíneo no cérebro. A concentração de sangue revela as áreas do cérebro envolvidas em uma tarefa cognitiva específica. Por exemplo, esta área é destacada quando a oxigenação aumenta, e logo após um período diminui. Esta abordagem possui resolução temporal baixa, mas possui alta precisão espacial, e apontou a ativação das áreas cerebrais envolvidas durante as tarefas de compreensão de código em pesquisas na área de engenharia de software.

4.7.1 Critérios da análise comparativa

Através das principais abordagens tradicionais utilizadas para medir a carga cognitiva nas pesquisas em engenharia de software foi possível identificar cinco principais critérios para caracterizar uma abordagem para medir a carga cognitiva em tarefas de compreensão de código.

C1. Resolução temporal: é importante que a abordagem possa refletir instantaneamente a carga cognitiva e para isso, requer alta resolução temporal.

C2. Compreensão de código: a abordagem deve ser específica e apropriada para tarefas de compreensão de código. Seja através de um processo de filtragem que isole as leituras de EEG ideais para o cálculo da carga cognitiva, que seja de interesse para tarefas de compreensão de código, ou de propriedades de compreensão de código para estabelecer relevância de áreas relacionadas a compreensão de código.

C3. Conectividade: as regiões do cérebro interagem durante tarefas cognitivas, tais como as tarefas de compreensão de código. Uma abordagem para medir a carga cognitiva durante tarefas de compreensão de código deve estabelecer conectividade de áreas que evidenciam interação durante essa tarefa.

C4. Independente: algumas abordagens dependem de um período de descanso antes de ser utilizada. Enquanto essa característica pode ser útil para experimentos controlados, em aplicações reais exigiria que os desenvolvedores de software descansassem antes de cada tarefa cognitiva, o que não torna a abordagem prática.

C5. Abordagem *over-time*: algumas abordagens dependem de várias da coleta de dados

cerebrais de múltiplas amostragens para viabilizar o uso da abordagem, este tipo de abordagem é conhecido como over-trial. Em contraste, uma abordagem para ser prática, e ter viabilidade para ser utilizada em ambientes reais, deve mensurar a carga cognitiva de modo *over-time*, isto é, deve refletir a carga instantânea com base no tempo (*time windows*).

Tabela 25: Análise comparativa entre as abordagens tradicionais utilizadas pela literatura de engenharia de software para mensurar a carga cognitiva.

Abordagens	C1. Resolução Temporal	C2. Compreensão	C3. Conectividade	C4. Independente	C5. Abordagem *Over-time*
CogEff	+	+	+	+	+
***Band power* Alfa**	+	-	-	+	+
***Band power* Beta**	+	-	-	+	+
***Band power* Delta**	+	-	-	+	+
***Band power* Theta**	+	-	-	+	+
ERD	+	-	-	-	+
ASR	+	-	-	+	+
PLV	+	-	+	-	-
BOLD	-	+	-	-	+

Legenda: + Contém - Não contém ∼ Contém parcialmente

Fonte: Elaborado pelo autor.

Em geral, através das lacunas apresentadas nas abordagens na Tabela 25, houve a possibilidade para propor uma abordagem para mensurar a carga cognitiva em tarefas de compreensão de código. Por isso, este capítulo buscou implementar uma abordagem que possua alta resolução temporal, forneça um processo de filtragem apropriado para mensurar a carga cognitiva em tarefas de compreensão de código, que estabeleça conectividade entre canais, que forneça propriedades para definir a relevância da interação entre canais durante as tarefas de compreensão de código, que não dependa de períodos de descanso, e seja *over-time*, que não dependa de coletar múltiplas amostragens para a mesma tarefa. Foi com base nesses critérios que a CogEff foi desenvolvida.

4.8 Considerações finais

Este capítulo apresentou a abordagem CogEff a qual mede a carga cognitiva de desenvolvedores de software em tarefas de compreensão de código baseado em dados EEG. Esta abordagem consiste em três etapas para calcular a carga cognitiva a partir dos dados EEG de desenvolvedores: A primeira etapa consiste em realizar a filtragem dos dados EEG, em seguida a conectividade entre os canais EEG é estabelecida em relação a esses dados, e finalmente, a CogEff calcula o valor final resumido o grafo através do cálculo da densidade. Os principais diferenciais da CogEff em relação as abordagens já utilizadas na literatura são: (1) a adoção de um processo de filtragem de dados EEG. Esse filtro foi adotado, pois apresentou ser eficaz para classificar compreensão de código baseados em dados EEG em um estudo anterior (GONÇALES et al., 2021); (2) utiliza uma abordagem *over-time* para estabelecer conexões entre canais EEG. Esses canais podem ser personalizados com pesos para quantificar a contribuições dessas conectividades em relação à compreensão de código. Esses pesos são definidos de forma intuitiva; (3) A CogEff resume o valor do grafo utilizando um cálculo de densidade. Portanto é uma

abordagem que busca quantificar a conectividade gerada durante as atividades de compreensão. Dessa forma, a CogEff busca ser uma abordagem relacionada à compreensão de código.

Para avaliar a correlação da CogEff com aspectos de tarefas de compreensão de código, o próximo capítulo realiza a correlação da CogEff com aspectos relacionados às tarefas de compreensão de código, tais como, as respostas das tarefas de compreensão, e o tempo decorrido para completar tarefas de compreensão. As abordagens utilizadas na literatura de engenharia de software também foram avaliadas.

5 AVALIAÇÃO DA COGEFF

Este capítulo busca responder à questão de pesquisa 3: "Qual a correlação das abordagens de carga cognitiva baseadas em dados EEG com a compreensão de código?". Esta questão de pesquisa busca resolver a falta de avaliação de correlação de indicadores de EEG que foram associados a carga cognitiva em estudos na engenharia de software. Este objetivo visa resolver o problema da falta de avaliações sobre a correlação das abordagens baseadas em EEG utilizadas na engenharia de software com tarefas de compreensão de código. Esta falta de avaliações ocasionou o risco de haver abordagens utilizadas não possuírem total potencial de serem relacionadas a tarefas de compreensão de código. Este capítulo visa em avaliar principalmente a abordagem CogEff pois foi proposta para ser específica para medir a carga cognitiva em tarefas de compreensão de código. A avaliação da abordagem CogEff, e os atuais abordagens utilizadas na literatura foi baseada em um *dataset* com dados EEG gerado para um experimento controlado de SEGALOTTO (2018). Este *dataset* contém dados de EEG de 33 participantes que executaram 10 tarefas de compreensão de código (SEGALOTTO, 2018), e foi produzido seguindo metodologias experimentais validadas na literatura (WOHLIN et al., 2012; FARIAS; GARCIA; LUCENA, 2014; FARIAS et al., 2015). Portanto, ao invés deste capítulo contribuir com a coleta de dados, esse capítulo contribui com o objetivo definido do estudo, variáveis, métodos de análise e avaliação dos resultados, resultados, e demais implicações.

Em particular, foi avaliado a correlação da CogEff com aspectos psicométricos relacionados a compreensão de código, tais como as respostas das tarefas de compreensão, e o tempo decorrido, em minutos, para concluir as tarefas de compreensão. Para isso, foram utilizados testes de correlação apropriados para o cenário de análise (WOHLIN et al., 2012; LEE et al., 2017), tais como o teste de *Kendall*, *Spearman* e *Pearson* (KENDALL, 1938; SPEARMAN, 1987; BENESTY et al., 2009). Essa avaliação é feita de forma comparativa com as abordagens tradicionais utilizados na literatura, tais como as frequências alfa (α), beta (β), delta (δ), gama (γ) e theta (θ), *Asymmetry Ratio* (ASR) e *Event-Related Desyncronization* (ERD) (LEE et al., 2017; KOSTI et al., 2018; CRK; KLUTHE, 2016). Além disso, foi avaliado a diferença dos valores da CogEff em tarefas de compreensão de código de complexidade diferente. A carga cognitiva foi representada pelo valor da CogEff.

5.1 Objetivo e questões de pesquisa

O objetivo deste estudo é analisar a abordagem CogEff, e abordagens tradicionais de EEG para mensurar a carga cognitiva em relação com aspectos das tarefas de compreensão de código, especificamente, com as respostas dos participantes, e o tempo gasto para concluir as tarefas de compreensão de código. Em particular, o objetivo desta avaliação é formulado usando o padrão GQM (*Goal Question Metric*) (CALDIERA; ROMBACH, 1994):

Analisar *as abordagens de EEG*

com o propósito de *investigar sua relação*
em respeito *as respostas, o tempo para concluir tarefas de compreensão, e nível de complexidade de código*
a partir do ponto de vista *de pesquisadores e desenvolvedores*
no contexto de *tarefas de compreensão de código fonte.*

Três questões de pesquisa foram formuladas com a finalidade de explorar as diferentes facetas deste objetivo:

- **QP 1**: a abordagem CogEff e as abordagens tradicionais de EEG de carga cognitiva possuem correlação com as respostas das tarefas de compreensão de código?
- **QP 2**: a abordagem CogEff e as abordagens tradicionais de EEG de carga cognitiva possuem correlação com o tempo para concluir tarefas de compreensão de código?
- **QP 3**: os valores da CogEff obtidos dos desenvolvedores em tarefas de complexidade maior são significativamente maiores em relação a tarefas de complexidade menor?

5.2 Hipóteses

Baseado nas questões definidas na seção 5.1 esta avaliação elabora três principais hipóteses que são descritas a seguir.

Hipótese 1: Pesquisas em engenharia de software e neurociência tem demonstrado, que o esforço mental tem refletido alterações na atividade oscilatória das ondas cerebrais (CRK; KLUTHE, 2016; KLIMESCH, 1999). Por exemplo, a abordagem ERD tem tendencia de aumentar ao passo que desenvolvedores executam tarefas de compreensão que não estão dentro do escopo de conhecimento dos participantes. Por outro lado, KLIMESCH (1999) alegaram que a banda alfa dos sinais EEG tem um comportamento inverso a banda theta, apresentando decrescimento na oscilação quando estão em tarefas de alta demanda. Mesmo com esse comportamento diferente dessas abordagens, elas foram utilizadas e adaptadas sem a avaliação da correlação de tais abordagens com compreensão de código. Dado que a abordagem CogEff utiliza sinais EEG para mensurar a carga cognitiva, mas com propriedades e aspectos que o torna específico para tarefas de compreensão de código, esta hipótese assume que a CogEff representa um comportamento coerente de esforço cognitivo aplicado durante tarefas de compreensão de código.

Por isso, esta hipótese conjectura que quanto menor o esforço cognitivo indicada pela diminuição do valor da CogEff, que é especificamente baseada nas abordagens de EEG (indicEEG), maior é a correlação com as respostas corretas de tarefas de compreensão de código. Desta forma, havendo uma correlação (Cor) negativa entre carga cognitiva e as respostas de compreensão de código, onde, enquanto o valor da CogEff diminui, indicando menor esforço mental, ao passo que as respostas das tarefas de compreensão tendem a ser corretas. No caso contrário,

se conjectura a falta dessa correlação, ou uma correlação positiva indicando de que a CogEff acompanha as respostas corretas das tarefas de compreensão. Esta relação não seria coerente com o esforço, pois o valor da CogEff iria aumentar ao passo que a quantidade de respostas corretas também aumenta, ou não apresentaria correlação com tarefas de compreensão caso a coeficiente de correlação for zero. Além disso, são avaliadas juntamente com a CogEff as demais abordagens baseadas em EEG para investigar assuntos relacionados a compreensão de código, tais como, as abordagens alfa (α), beta (β), delta (δ), theta (θ), ASR e ERD.

Hipótese Nula 1, H_{1-0}: A correlação das abordagens cognitivas baseados em EEG e a as respostas das tarefas de compreensão de código é maior ou igual a zero.

H_{1-0}: $Corr(IndicEEG, corretude) \geq 0$

Hipótese Alternativa 1, H_{1-1}: A correlação entre indicadores cognitivos baseados em EEG e a taxa de respostas corretas em tarefas de compreensão de código é menor que zero.

H_{1-0}: $Corr(IndicEEG, corretude) < 0$

Hipótese 2: O tempo para completar atividades instrucionais também é um fator que pode indicar a carga cognitiva durante a manipulação de artefatos de trabalho (SWELLER, 2010). No ramo de psicologia cognitiva, o tempo é frequentemente utilizado para relacionar carga cognitiva a compreensão de materiais instrucionais (SWELLER, 2010; CRK; KLUTHE, 2016). Especificamente, materiais que impõem menor carga cognitiva exigem uma memória de trabalho menor para resolver exige menos tempo, e caso a tarefa exigir maior carga cognitiva e mais memória de trabalho, mais tempo exige para completar atividades instrucionais.

Desse modo, espera-se que a CogEff, abordagem para mensurar a carga cognitiva em tarefas de compreensão de código, obtenha uma correlação positiva em relação ao tempo decorrido para resolver a tarefa de compreensão de código. Isso significa que, ao passo que a tarefas de compreensão demandarem mais tempo para serem resolvidas, maior será o valor da CogEff. Isto também significa que, ao passo que as tarefas de compreensão demandarem menos tempo para serem resolvidas, menor deverá ser o valor da CogEff. Além disso, também se espera o mesmo comportamento das abordagens baseadas em EEG para mensurar a carga cognitiva na engenharia de software tais como, a frequência alfa (α), beta (α), delta (δ), theta (θ), as abordagens *Assymetry Ratio* (ASR), e *Event-Related Desynchronization* (ERD). Essas abordagens baseadas em EEG serão analisados em relação a CogEff, porque tem sido utilizado como maneira direta para medir a carga cognitiva, ao invés da utilização de aspectos psicométricos (ANTONENKO et al., 2010).

Hipótese Nula 2, H_{2-0}: A correlação entre indicadores cognitivos baseados em EEG e o tempo de resposta em tarefas de compreensão de código é menor ou igual a zero.

H_{2-0}: $Corr(IndicEEG, tempo) \leq 0$

Hipótese Alternativa 2, H_{2-1}: A correlação entre indicadores cognitivos baseados em EEG e o tempo de resposta em tarefas de compreensão de código é maior que zero.

H_{2-1}: $Corr(IndicEEG, tempo) > 0$

Hipótese 3: Pesquisas em engenharia de software apontam a relação de aspectos de má qua-

lidade de código com carga cognitiva (FAKHOURY et al., 2018; YEH et al., 2017; MÜLLER; FRITZ, 2016). É de senso comum na indústria de software considerar associar código com alta complexidade a aspectos de má qualidade que consequentemente prejudica a compreensão e a manutenibilidade de software (TROCKMAN et al., 2018). Faltam evidências em estudos controlados, de que uma complexidade alta impacta na carga cognitiva dos desenvolvedores de software. Deste modo, espera-se que, quanto maior a complexidade ciclomática de um código fonte, maior é a imposição de carga cognitiva nos usuários. Do modo contrário, quanto menor a complexidade ciclomática, menor é a imposição de carga cognitiva nos participantes.

Especificamente, a hipótese alternativa assume que um conjunto de tarefas de compreensão com nível de complexidade maior (*complMaior*) exigem mais esforço mental dos participantes em relação às tarefas de compreensão que possuem um nível de complexidade menor (*complMenor*). Pois, a indústria assume que uma tarefa com complexidade maior prejudica a compreensão de software (SCALABRINO et al., 2019; TROCKMAN et al., 2018; SCALABRINO et al., 2017). Dessa forma, a hipótese nula assume que, uma tarefa de compreensão de código com complexidade maior (*complMaior*) não tem impacto na carga cognitiva aplicado em relação a uma tarefa de compreensão com complexidade menor (*complMenor*).

Hipótese Nula 3, H_{3-0}: Código fonte com complexidade menor (*ComplMenor*) exige uma carga cognitiva maior ou igual que código fonte com complexidade maior (*ComplMaior*).

H_{3-0}**:** $CargaCognitiva(\textbf{\textit{Código}})_{ComplMenor} \geq CargaCognitiva(\textbf{\textit{Código}})_{ComplMaior}$

Hipótese Alternativa 3, H_{3-1}: Código fonte com complexidade menor (*ComplMenor*) exige uma carga cognitiva menor que código fonte com complexidade maior (*ComplMaior*).

H_{3-1}: $CargaCognitiva(\textbf{\textit{Código}})_{ComplMenor} < CargaCognitiva(\textbf{\textit{Código}})_{ComplMaior}$

5.3 Perfil dos participantes

No conjunto de dados utilizado neste trabalho, o qual foi produzido para um experimento controlado no estudo de SEGALOTTO (2018), foram recrutados um total de 35 participantes com idade que varia entre 17 e 46 anos.

Educação: alguns participantes são graduados e participam do curso de pós-graduação em computação aplicada (31%, 11/35). Cerca de 29% (10/35) completaram o ensino superior. Aproximadamente 11% dos participantes completaram o ensino médio (4/35), 5% (2/35) dos participantes estavam dedicados ao programa de doutorado em computação aplicada, 2% finalizaram o mestrado em computação aplicada (1/35), e 2% tinham especialização (1/35). Um participante cursa ensino médio (1/35). O curso de graduação dos participantes é relacionado ao desenvolvimento de software. Especificamente, eles estão envolvidos em cursos como ciência da computação (29%, 10/35), sistemas de informação (17%, 6/35), análise de sistemas (14%, 5/35), jogos digitais (14%, 5/35), entre outros cursos (26%, 7/35) tais como engenharia da computação, engenharia elétrica, segurança da informação, e curso técnico em manutenção de computadores.

Profissão: Aproximadamente 34% (12/35) dos participantes trabalham principalmente como desenvolvedores de software, e 23% (8/35) são assistentes de pesquisa. Cerca de 14% (5/35) trabalham como analistas de software, 3% (1/35) como assistentes de computadores, 2% (1/35) são cientistas de dados, 2% (1/35) são gerentes de projeto, 2% (1/35) são engenheiros de software. Os participantes também são empreendedores (2%, 1/35), técnicos em informática (2%, 1/35), e cerca de 11% (4/35) não especificaram seus cargos. um total de 34% (12/35) trabalham nas universidades. Além disso, os participantes também trabalham em diferentes empresas tais como Banrisul, Correios, GetNet, Altus, CIGAM, Kenta, Ilegra e Umbler. Cerca de 14% (5/35) dos participantes não revelaram o nome do local de trabalho.

Habilidades: Cerca de 57% (20/35) dos participantes têm até 2 anos de experiência com desenvolvimento de software. Aproximadamente 14% (5/35) dos participantes desenvolvem software entre 3 e 4 anos, 9% (3/35) dos participantes trabalham com desenvolvimento de software entre 5 e 6 anos, e 20% (7/35) possuem mais de sete anos de experiência com desenvolvimento de software. Em relação linguagem Java, isto é, a linguagem utilizada nas tarefas deste estudo, 46% (16/35) dos participantes reportaram que possuem conhecimentos básicos sobre a linguagem, 26% (9/35) conhecem moderadamente a linguagem, e 29% (10/35) se consideram proficientes na linguagem Java.

Formulário de consentimento: Antes de executar as tarefas experimentais, os participantes concordaram com um termo de consentimento autorizando que seus dados psicofisiológicos sejam utilizados para a finalidade de pesquisa.

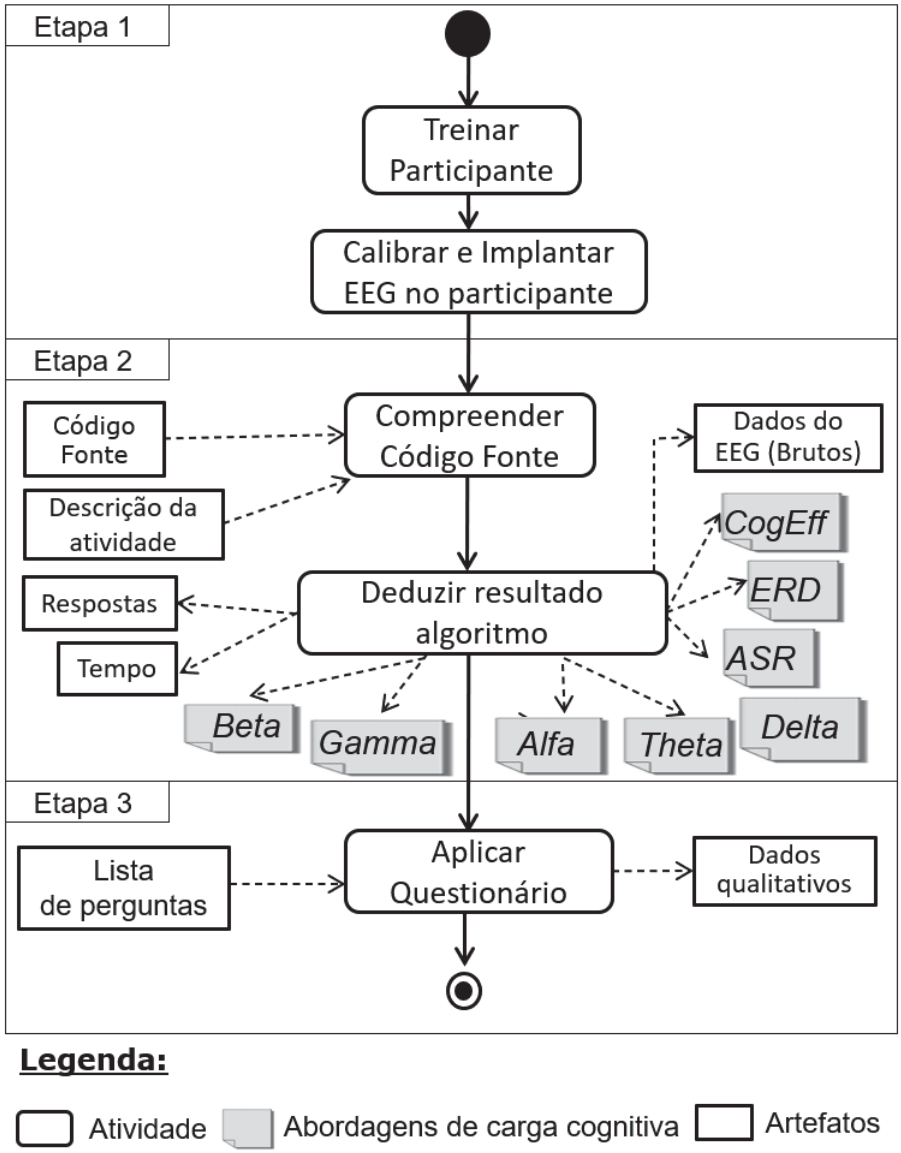
5.4 Processo experimental

Esta Seção descreve parte do processo experimental derivado do *dataset* utilizado em SEGALOTTO (2018). Especificamente as etapas 1 e 3 são as mesmas utilizadas em SEGALOTTO (2018). A etapa 2 muda pois foram utilizadas outras variáveis a partir do EEG tais como as ondas de frequência alfa (α), beta (β), delta (δ), gamma (γ), theta (θ), ASR e abordagem CogEff. Além disso, esta tese utilizou especificamente a etapa 2 para aplicar as abordagens de EEG, e etapa 3 para analisar o perfil dos participantes. A Figura 16 apresenta esse processo experimental organizado em três etapas, incluindo a etapa de onde as abordagens utilizada nesta tese foram extraídas.

Etapa 1. Esta etapa consiste no início do experimento para os desenvolvedores de software. **Treinamento.** Todos os participantes foram treinados em relação ao tipo de tipo de tarefas de compreensão e também receberam um formulário de consentimento com a realização das atividades experimentais e finalidade do uso da pesquisa. **Calibrar e implantar sensores EEG.** Configurar e implantar o EEG para capturar ondas cerebrais dos participantes.

Etapa 2. Nesta etapa os participantes executam as tarefas de compreensão. **Compreender código-fonte.** Primeiro, os desenvolvedores se concentram em analisar e entender o código-fonte. Cada participante realizou 10 tarefas de compreensão de código. Os participantes têm

Figura 16: Visão geral do fluxo de processamento dos dados EEG.



Fonte: Elaborado pelo autor.

um limite de 60 segundos para executar uma tarefa de compreensão de código. A tarefa de compreensão consiste no desenvolvedor analisar, integrar o significado das instruções do código fonte. Nesta etapa, a entrada são o código fonte, e a descrição da atividade. **Deduzir resultado do algoritmo.** Ao fim da tarefa o desenvolvedor deduz mentalmente o resultado baseado no raciocínio desenvolvido durante a compreensão, e escolhe uma resposta entre as cinco opções apresentadas. A saída dessa etapa é uma gama de indicadores psicométricos e psicofisiológicos utilizados para análise nesta etapa da pesquisa. Os resultados psicométricos dessa etapa são o *tempo* gasto pelo usuário para realizar a tarefa, e as *respostas*. A partir das respostas é calculado a taxa de repostas corretas. As saídas psicofisiológicas são os *dados brutos* das ondas cerebrais (EEG) relacionado a toda tarefa de compreensão. A partir desses dados brutos, são extraídas as demais abordagens, tais como, a *CogEff*, e os demais utilizados pela literatura, tais como,

Event-Related Desynchronization (ERD), *Assimmetric Ratio* (ASR), e as frequências alfa (α), beta (β), delta (δ), e theta (θ).

Etapa 3. Na última etapa os participantes forneceram informações sobre seu perfil profissional. **Responder Questionário.** Um questionário foi fornecido a cada participante para coletar dados pessoais tais como formação acadêmica, cargo profissional, tempo de estudos em universidades, experiência com linguagens de programação, e experiência profissional. A saída dessa tarefa foram os dados qualitativos para descrever o perfil dos participantes nesse experimento.

5.5 Especificação das tarefas de compreensão

No estudo de SEGALOTTO (2018) participantes executaram 10 tarefas de compreensão de código. A Tabela 26 apresenta a descrição dessas tarefas de compreensão. Especificamente, a Tabela 26 descreve os principais aspectos dessas tarefas, tais como, o ID, o contexto para o qual o trecho de código foi elaborado, a quantidade de linhas, a quantidade de classes, quantidade de métodos e a linguagem de programação a qual o trecho de código está expressa.

Tabela 26: Descrição das tarefas de compreensão.

ID	Contexto da Tarefa	#Linhas	#Comp. Ciclomática	#Classes	#Métodos	Linguagem
T1	Processar Salário	10	1	1	1	Java
T2	Processar Salário v2	17	5	1	5	Java
T3	Processar Pontos	8	1	1	1	Java
T4	Processar Pontos v2	14	2	2	2	Java
T5	Obter Calendário	6	1	1	1	Java
T6	Obter Calendário v2	30	5	5	5	Java
T7	Contador Elementos Fila	10	2	1	1	Java
T8	Contador Elementos Fila v2	16	3	2	2	Java
T9	Contador de Cartões	19	5	1	1	Java
T10	Contador de Cartões v2	44	9	5	5	Java
Média		17,4	3,4	2	2,4	-

Fonte: Elaborado pelo autor.

Os participantes executaram 10 tarefas com uma média de 10,7 linhas por tarefa. Além disso, cada tarefa de compreensão continha tinha em média 2 classes, e 2,4 métodos por tarefa. Todas as tarefas consistiam em trechos de código na linguagem Java. Além disso, as tarefas consistem em duas versões da mesma funcionalidade, onde a segunda versão foi acrescida mais métodos e linhas em relação a primeira versão.

5.6 Variáveis

Esta seção apresenta as variáveis dependentes e independentes em relação as hipóteses definidas na Seção 5.2. Essas variáveis são exclusivas deste estudo.

A hipótese 1 investiga a relação entre indicadores psicofisiológicos e taxa de repostas corretas. ***Variáveis independentes:*** indicadores baseados em ondas cerebrais tais como o indicador proposto nesse trabalho, *CogEff*, o *Event Related Desynchronization* (ERD), *Assymetry Ratio*

(ASR), e a potência de sinal das frequências alfa (α), beta (β), theta (θ), e delta (δ) de canais EEG que se concentram no lobo frontal do hemisfério esquerdo. ***Variáveis dependentes:*** A variável dependente são as respostas das questões.

A hipótese 2 investiga a relação entre indicadores psicofisiológicos e tempo médio para completar as tarefas. *Variáveis independentes:* as mesmas da hipótese 1, tanto para indicadores psicofisiológicos quanto as métricas de código fonte. *Variáveis dependentes:* A variável dependente é o tempo em minutos para responder as tarefas de compreensão de código.

A hipótese 3 investiga o impacto que tarefas de compreensão de código de complexidade maior causa no esforço cognitivo nos desenvolvedores em relação a tarefas de complexidade menor. *Variáveis independentes:* Nível de complexidade das tarefas definidas pela quantidade de linhas e complexidade ciclomática. Especificamente, as tarefas foram classificadas em duas categorias de complexidade. As tarefas de complexidade menor (*complMenor*), e de complexidade maior (*complMaior*). As tarefas de complexidade menor (*complMenor*) são as que contém complexidade ciclomática inferior à média 3. As tarefas de complexidade maior (*complMaior*) contém complexidade ciclomática maior ou igual a 3. *Variáveis dependentes:* as variáveis dependentes, que vão representar o esforço cognitivo de desenvolvedores de software, são os indicadores CogEff, e o ERD.

5.7 Procedimentos de análise

Para a condução das análises das três hipóteses formuladas nesse trabalho procedimentos para descrever os dados estatisticamente e testes de hipóteses foram definidos. Esses procedimentos são exclusivos deste trabalho e são descritos a seguir.

Estatística Descritiva. Será apresentado o resumo dos dados através da estatística descritiva. Especificamente, dados são todas as variáveis psicofisiológicas e métricas de código utilizadas nesse estudo. Esta etapa da análise também verifica a distribuição dos dados, isto é, se os dados são normalmente distribuídos foram aplicados os testes de *Kolmogorov Smirnov* e *Shapiro Wilk* (DEVORE; FARNUM; DOI, 2013). Ambos os testes foram conduzidos para satisfazer os aspectos do tamanho da amostragem. Por exemplo, *Shapiro Wilk* é apropriado para amostragens menores. Ambos os testes de distribuição rejeitam a normalidade se $p - value < 0.05$.

Teste de hipóteses. Com a finalidade de correlacionar duas populações nas hipóteses 1 e 2 é primeiro verificado se é adequado aplicar o teste paramétrico da correlação de *Pearson*, caso o teste de distribuição apontar normalidade dos dados. Caso contrário, é aplicado o teste de correlação não paramétrico de *Spearman*. A hipótese 3, que investiga as mesmas populações em tratamentos diferentes (tarefas de complexidade aumentada e complexidade reduzida) conduz testes de hipóteses pareados. Neste caso, também é verificado a aplicabilidade do *t-test*, caso seja confirmada a normalidade desses dados. Ainda assim, a análise é repetida com testes não paramétricos, especificamente com o teste de Wilcoxon. Ambos os testes rejeitam a hipótese nula H_{3-0} com $p - value < 0.05$.

5.8 Resultados

Esta seção apresenta os resultados dos testes para as hipóteses formuladas. A Seção 5.8.1 apresenta a estatística descritiva das questões de pesquisas formuladas na metodologia. A Seção 5.8.2 apresenta os resultados sobre a correlação das abordagens de EEG com as repostas das tarefas de compreensão. A Seção 5.8.3 apresenta os resultados sobre a correlação das abordagens de EEG com o tempo para concluir tarefas de compreensão. A Seção 5.8.4 apresenta os resultados sobre diferença do esforço entre tarefas de complexidade reduzida e aumentada, baseado no indicador da CogEff.

5.8.1 Estatística descritiva

Estatística descritiva correlação dos indicadores psicofisiológicos com as respostas (QP1) e o tempo (QP2). A Tabela 27 apresenta um resumo dos valores dos indicadores psicofisiológicos obtidos durante as tarefas de compreensão. Os indicadores psicofisiológicos se referem a abordagem desenvolvida nesta pesquisa, CogEff, e as demais abordagens de EEG utilizadas nas pesquisas em engenharia de software para investigar compreensão de software, especificamente a onda alfa, theta, beta, delta, o Event Related Desyncronization (ERD) e Asymmetry Ratio (ASR). Os valores das métricas foram normalizados entre valores de 0 e 1. Os valores foram normalizados para representação e análise dos dados em uma escala semelhante.

Tabela 27: Estatística descritiva das variáveis psicofisiológicas.

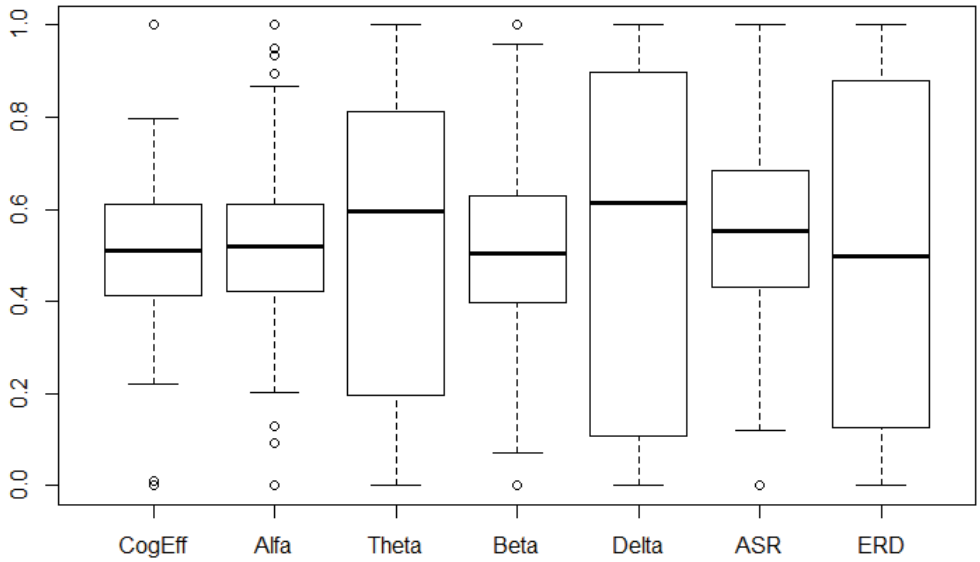
Métricas	N	Min.	25	Mediana	Média	75	Máximo	Desvio Padrão	Variância
CogEff	350	0	0.411	0.51	0.511	0.612	1	0.12	0.014
Alfa	350	0	0.422	0.519	0.518	0.611	1	0.13	0.01
Theta	350	0	0.197	0.594	0.512	0.812	1	0.31	0.101
Beta	350	0	0.396	0.505	0.514	0.630	1	0.142	0.02
Delta	350	0	0.108	0.613	0.502	0.898	1	0.390	0.152
ERD	350	0	0.125	0.499	0.497	0.879	1	0.370	0.136
ASR	350	0	0.432	0.552	0.561	0.683	1	0.20	0.04

Fonte: Elaborado pelo autor.

A Figura 17 apresenta a distribuição visual dos dados da Tabela 27. Ambos os testes de normalidade *Kolmogorov* e *Shapiro-Wilk* rejeitam a hipótese de que os dados não pertençam a uma distribuição normal. Desta forma, tem-se a preferência de executar testes paramétricos para o teste de correlação, tais como o teste de *Pearson*. Desta forma, para testar a correlação das abordagens de EEG com o tempo, foi executado o teste de *Pearson*, e o teste não paramétrico de *Spearman* foi repetido para reforçar os resultados. Na primeira hipótese, h1, foi executado apenas testes não paramétricos, porque a variável dependente de respostas é categórica. Desta forma, ambos os testes de *Spearman* e *Kendall* foram executados.

Estatística descritiva sobre a diferença de esforço em tarefas com complexidade aumentada e reduzida (QP3). A Tabela 28 descreve a distribuição dos valores da CogEff entre

Figura 17: Representação da distribuição dos dados.



Fonte: Elaborado pelo autor.

as duas categorias de tarefas de compreensão, isto é, com complexidade aumentada e complexidade reduzida. Essas tarefas são as especificadas na Tabela 26 da Seção 5.5. As tarefas de complexidade aumentada são as tarefas 2, 6, 9 e 10 e as tarefas classificadas como de complexidade reduzida são as tarefas 1, 3, 4, 5 e 7. Como a quantidade de tarefas entre os grupos é desigual, consequentemente a quantidade de amostras (N) na Tabela 28 do grupo de tarefas de complexidade reduzida (778) é maior em relação ao número de amostras do grupo de tarefas de complexidade aumentada (614).

Tabela 28: Estatística descritiva da CogEff nas tarefas de diferentes níveis de complexidade.

Métricas	**N**	**Min.**	**25**	**Mediana**	**Média**	**75**	**Maximo**	**Desvio Padrão**	**Variância**	**t-test**	**wilcoxon**
Comp. Reduzida	778	0	0.72	0.75	0.72	0.78	1	0.17	0.02	0.005	<0.0001
Comp. Aumentada	614	0	0.74	0.781	0.748	0.80	1	0.16	0.02		

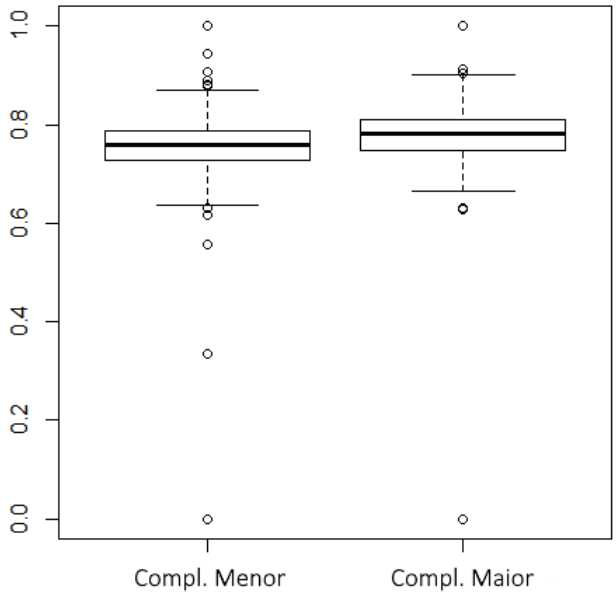
Fonte: Elaborado pelo autor.

A Figura 18 apresenta a distribuição visual dos dados apresentados na Tabela 28. Os testes de anderson darling e kolmogorov smirnov rejeitaram a hipótese de que os dados pertencerem a uma distribuição normal. Além disso, os dados aparentemente apontam que o esforço mental indicado pela abordagem CogEff é superior no grupo de tarefas de complexidade aumentada, em relação as tarefas de complexidade reduzida. Os resultados são discutidos na Seção 5.8.4.

5.8.2 QP1: Resultados da Hipótese 1 - correlação das abordagens com as respostas das tarefas de compreensão de código

A Tabela 29 apresenta os resultados da correlação entre as abordagens de EEG utilizadas para mensurar a carga cognitiva e as respostas das tarefas de compreensão. Especificamente,

Figura 18: Representação da distribuição dos dados da Hipótese 3.



Fonte: Elaborado pelo autor.

os resultados da Tabela 29 se referem a correlação de *Kendall*. A Tabela 29 mostra os valores do *p-value*, e os valores de correlação. Valores do *p-value* menores que 0,05 representam que a correlação é significativa. A correlação pode variar de -1 a 1, onde valores negativos significam correlação inversa, isto é, os valores seguem direção opostas as respostas corretas. E correlação positiva significa uma concordância entre as abordagens baseadas em EEG para mensurar a carga cognitiva e as respostas corretas das tarefas de compreensão de código.

Os resultados da correlação de *Kendall* na Tabela 29 mostram que, no geral, a CogEff possui uma correlação negativa com as respostas das tarefas de compreensão de código. Além disso, a correlação é significativa com *p-value* < 0,001, validando a hipótese alternativa 1 (H_{1-1}). Este tipo de correlação é esperado para refletir um aumento da carga cognitiva, pois STIPACEK et al. (2003) relacionaram o aumento da memória de trabalho (*working memory*), associada a carga cognitiva, ao aumento da dessincronização dos sinais EEG durante tarefas de memorização. A correlação negativa de -0.2893 aponta que resultados mais altos da CogEff estão relacionados a respostas incorretas e valores menores da CogEff estão correlacionados a repostas corretas.

No geral, a CogEff obteve uma correlação negativa em quase todas as tarefas, em exceção da tarefa 4. Essa correlação não foi significativa nas tarefas de compreensão 2, 4, 6, 7, 8, 9 e 10, isto é, o *p-value* resultou em valores acima de 0,05. Por outro lado, a correlação da CogEff foi significativa nas tarefas 1, 3 e 5. Os motivos desses resultados especificamente estão relacionados a fatores humanos. O primeiro é que provavelmente os desenvolvedores poderiam ter certeza coerente em resolver as tarefas 1, 3, e 5. O segundo motivo, é que nas demais tarefas, alguma parte dos desenvolvedores responderam incorretamente uma tarefa que acreditavam serem simples. O cenário contrário também seria possível, uma tarefa fácil pode impor carga cognitiva elevada por causa da impressão equivocada por parte dos desenvolvedores. A predo-

minância de ocorrência desses dois cenários pode ter sido a causa da tarefa 4 resultar em uma correlação próxima a zero.

Tabela 29: Resultado da correlação de *Kendall* na hipótese 1

Tarefas	Estatísticas	Abordagens EEG							Corretude
		Alfa	Theta	Beta	Delta	ERD	ASR	CogEff	
1	**Correlação**	**0,3171**	0,0935	**0,3587**	0,1403	-0,1923	0,2694	**-0,3245**	73,9%
	p-value	**0,0258**	0,5109	**0,0117**	0,3241	0,1766	0,1891	**0,0247**	
2	**Correlação**	0,1044	-0,0237	0,1091	-0,0711	-0,1187	-0,0133	-0,1499	36,9%
	p-value	0,4633	0,8676	0,4432	0,617	0,4045	0,9433	-0,2083	
3	**Correlação**	**0,3508**	-0,2033	0,3152	-0,1728	0,2033	-0,1576	**-0,3448**	68,8%
	p-value	**0,0141**	0,1548	0,02745	0.2266	0,1548	0,2702	**0,0174**	
4	**Correlação**	0,2358	0,1232	-0,2197	-0,0750	-0,0321	0,24655	0,0422	75%
	p-value	0,0967	0,3854	0,1218	0,5972	0,8208	0,08254	0,7697	
5	**Correlação**	**0,3355**	-0,1917	**0,3523**	0,1006	-0,0623	0,3402	**-0,3202**	56,5%
	p-value	**0,0184**	0,1779	**0,0379**	0,4794	0,6615	0,1074	**0,0267**	
6	**Correlação**	-0,1148	-0,1722	-0,1200	0,1775	-0,0104	-0,0300	-0,1811	30,5%
	p-value	0,4214	0,2278	0,4006	0,214	0,9417	0,8825	0,2119	
7	**Correlação**	0,2357	-0,1980	-0,0565	-0,1225	-0,0094	**0.2828**	-0,1131	44,1%
	p-value	0,0977	0,1642	0,691	0,3892	0,9472	**0,0469**	0,4267	
8	**Correlação**	**0,3305**	-0,0814	**0,3257**	-0,0574	**-0,4358**	0,3060	-0,2107	40,1%
	p-value	**0,0201**	0,567	**0,0220**	0,6861	**0,0021**	0,0737	0,1384	
9	**Correlação**	0,1872	0,1435	-0,1872	-0,0249	**0,2873**	0,2027	-0,2484	9,7%
	p-value	0,189	0,3139	0,189	0,861	**0,04399**	0,2819	0,0859	
10	**Correlação**	0,1416	0,0708	0,1133	0,0991	0,0708	0,3010	-0,2000	2,6%
	p-value	0,3217	0,6202	0,4279	0,4878	0,6202	0,146	0,1684	
Todas	**Correlação**	**0,2600**	-0,0427	**0,2748**	-0,0245	-0,0640	**0,0960**	**-0,2893**	43,8%
	p-value	**< 0,001**	0,3293	**< 0,001**	0,5751	0,1435	**0,0282**	**< 0,001**	

Fonte: Elaborado pelo autor.

Outra observação é que a CogEff também se comporta como uma abordagem que mensura a carga cognitiva porque os resultados obtidos são de comportamento inverso a outras abordagens utilizados para mensurar carga cognitiva nos estudos de engenharia de software. Apesar da onda Alfa (α) e Beta (β) obtiveram correlação positiva com as respostas, o comportamento da correlação foi similar a abordagem CogEff. Estes resultados, reforçam a utilidade da CogEff como meio para mensurar a carga cognitiva em tarefas de compreensão de código.

A CogEff possui 4 principais diferenciais em relação às abordagens existentes, tais como: áreas de compreensão: a abordagem leva em consideração regiões cerebrais relacionadas a compreensão de código. As demais técnicas não suportam nenhum tipo de customização ou heurísticas para fornecer pesos às regiões que podem estar relacionadas a compreensão de código; Conectividade: a CogEff calcula a partir da conectividade de todos os sensores utilizados, e portanto, seu resultado resume o comportamento de todos os sensores, e engloba o comportamento de todas as regiões; *baseline*: a CogEff não necessita de nenhuma baseline para ser calculada, diferentemente do ERD, que necessita de um intervalo específico antes de cada tarefa para ser estimada; *trials*: não depende de *trials* da mesma tarefa, pois a CogEff utiliza uma abordagem *over-time* para estipular as conexões. Isso permite que as conexões sejam estipuladas baseadas em uma janela de tempo atual.

A Tabela 30 também mostra os *p-values*, e os valores de correlação, especificamente dos tes-

tes de *Spearman*. Os resultados dessa tabela confirmam os resultados anteriores. Em particular, a CogEff também possui uma correlação negativa, de -0.3537, com as respostas das tarefas de compreensão de código. Além disso, a correlação também é significativa, com *p-value* < 0,001, também validando a hipótese alternativa 1 (H_{1-1}). Os resultados da Tabela 30 também confirmam os resultados obtidos em relação as tarefas de compreensão, onde a CogEff obteve uma correlação negativa em quase todas as tarefas, em exceção da tarefa 4. Na Tabela 30, a correlação de *Spearman* também mostrou que não foi significativa nas tarefas de compreensão 2, 4, 6, 7, 8, 9 e 10, isto é, o *p-value* resultou em valores acima de 0,05. Por outro lado, a correlação da CogEff foi significativa nas tarefas 1, 3 e 5.

Tabela 30: Resultado dos testes de correlação de *Spearman* para a hipótese 1.

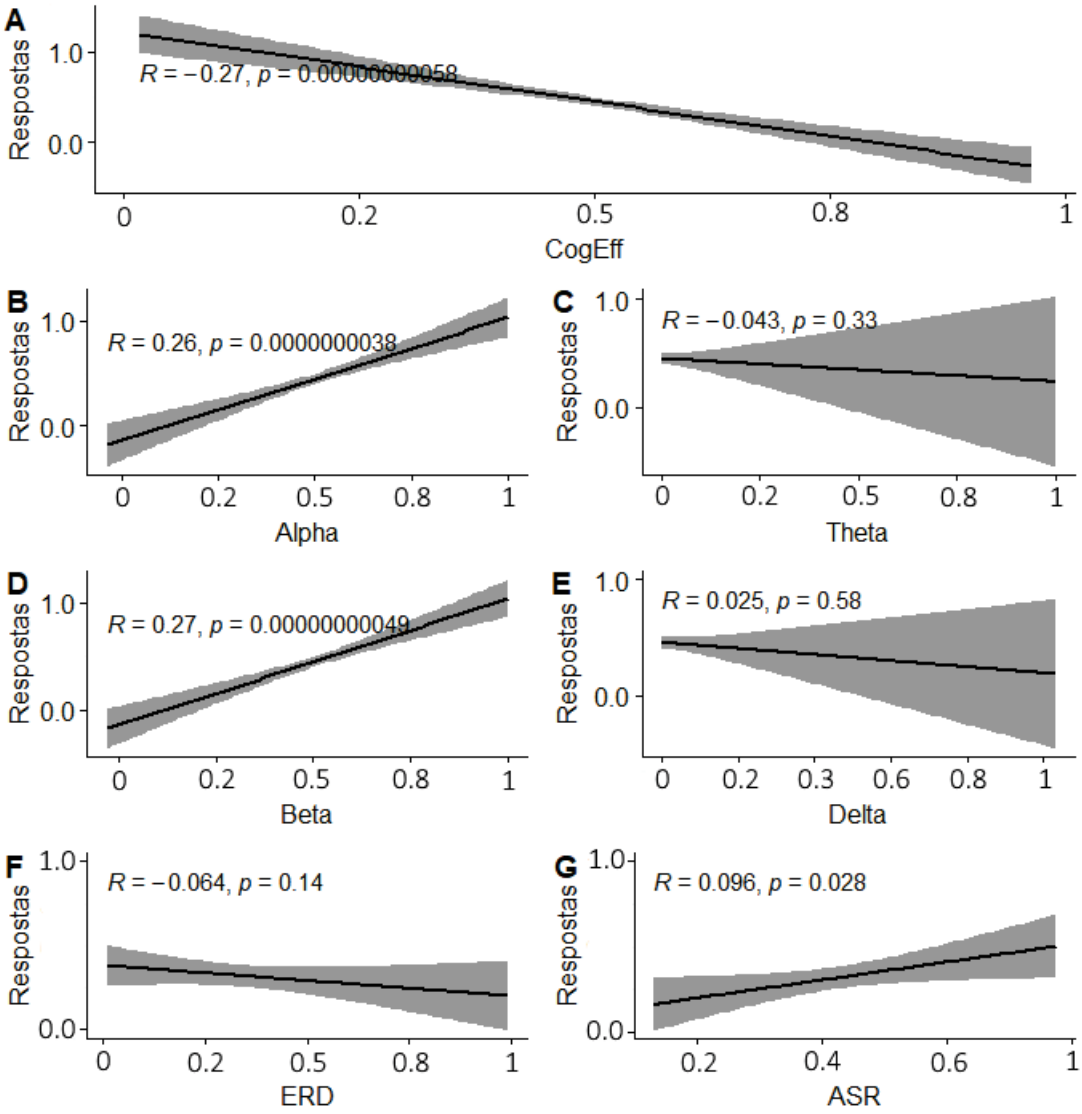
Tarefas	Estatisticas	Abordagens EEG							Corretude
		Alfa	Theta	Beta	Delta	ERD	ASR	CogEff	
1	**Correlação**	**0,3820**	0,1127	**0,4321**	0,1691	-0,2317	0,3185	**-0,3908**	73,9%
	p-value	**0,0235**	0,519	**0,0095**	0,3315	0,1804	0,1977	**0,0222**	
2	**Correlação**	0,1257	-0,028	0,1315	-0,0914	-0,1429	-0,0159	-0,2506	36,9%
	p-value	0,4715	0,8705	0,4514	0,6012	0,4126	0,9455	0,1528	
3	**Correlação**	**0,4208**	0,2439	**0,3781**	-0,2073	0,2439	-0,1890	**-0,4137**	68,8%
	p-value	**0,0118**	0,1578	**0,0250**	0,2319	0,1578	0,2767	**0,0150**	
4	**Correlação**	0,2848	0,1488	0,2653	-0,0906	-0,0388	0,2977	-0,0509	75%
	p-value	0,0972	0,3934	0,1234	0,6047	0,8247	0,0823	0,7747	
5	**Correlação**	**0,4043**	0,2310	**0,3523**	0,1212	-0,0750	0,4025	**-0,3856**	56,5%
	p-value	**0,016**	0,1818	**0,0379**	0,4876	0,6681	0,1092	**0,0242**	
6	**Correlação**	0,1378	0,2068	-0,1441	0,2130	-0,0125	-0,0358	-0,2173	30,5%
	p-value	0,4296	0,2332	0,4087	0,219	0,943	0,8877	0,217	
7	**Correlação**	0,2840	0,2385	-0,0681	-0,1476	-0,0113	**0,3408**	-0,1363	44,1%
	p-value	0,0982	0,1675	0,6972	0,3972	0,9484	**0.0451**	0,4349	
8	**Correlação**	**0,3985**	-0,0981	**0,3927**	-0,0693	**-0,5255**	0,3060	-0,3626	40,1%
	p-value	**0,0177**	0,5747	**0,01961**	0,6924	**0,0011**	0,0737	0,1392	
9	**Correlação**	0,2252	0,1727	0,2252	-0,0300	**0,3454**	0,2406	-0,2989	9,7%
	p-value	0,1932	0,3211	0,1932	0,864	**0,0421**	0,2934	0,0859	
10	**Correlação**	0,1699	0,0849	-0,1359	0,1189	0,0849	0,3526	-0,2397	2,6%
	p-value	0,329	0,6274	0,4361	0,496	0,6274	0,1512	0,172	
Todas	**Correlação**	**0,3179**	-0,0522	**0,3360**	0,0300	-0,0783	**0,1174**	**-0,3537**	43,8%
	p-value	**< 0,001**	0,33	**< 0,001**	0,5759	0,1437	**0,0280**	**< 0,001**	

Fonte: Elaborado pelo autor.

A Figura 19 apresenta a correlação de *Kendall* conforme os resultados da Tabela 29. Como afirmado anteriormente, a CogEff tem uma correlação negativa com as respostas, enquanto dois principais indicadores de esforço mental utilizado na literatura de engenharia de software, possui correlação positiva. A linha do teste de correlação da CogEff (A) está mais elevada na direção das repostas corretas nos valores menores da CogEff, indicando que, o menor esforço está correlacionado a respostas corretas, enquanto, o valor da CogEff aumenta, a linha tende a se aproximar das repostas erradas. As ondas Alfa (B) e Beta (D), obtiveram uma correlação significativa, mas, as linhas se aproximam das repostas corretas quando ambos os indicadores aumentam.

Além disso, o indicador de dessincronização ERD, que foi utilizado como uma abordagem de carga cognitiva na engenharia de software, tem um comportamento muito parecida com a

Figura 19: Representação da correlação de *kendall* entre abordagens EEG e respostas de todas as tarefas de compreensão de código.



Fonte: Elaborado pelo autor.

abordagem CogEff. Porém, nas tarefas de compreensão essa correlação não foi significativa. Isto pode ser uma consequência do design do experimento. Os indicadores theta (C) e delta (E) apresentaram uma linha menos inclinada e, portanto, mostrando nenhuma correlação com as repostas. Por fim, o indicador ASR possui um comportamento semelhante aos indicadores alfa (B) e beta (D). Os resultados indicam que CogEff é uma abordagem coerente com a carga cognitiva dos desenvolvedores nas tarefas de compreensão de código.

Conclusão QP1: *os resultados de correlação de Kendall e Spearman mostram que a CogEff possui correlação negativa em relação as repostas das tarefas de compreensão de código deste conjunto de dados. Esse resultado indica que, enquanto o valor da CogEff aumenta há a diminuição da quantidade de respostas corretas, e que o valor da CogEff diminui ao passo que a quantidade de repostas corretas aumenta. Por fim, essa correlação foi significativa, desta forma, rejeitando a hipótese nula H_{1-0} e confirmando a hipótese alternativa H_{1-1}.*

5.8.3 QP2: Resultados da Hipótese 2 - correlação das abordagens com o tempo decorrido para resolver tarefas de compreensão de código

Esta seção responde à questão de pesquisa 2 da avaliação da abordagem CogEff. Os resultados da Tabela 31 apresentam que as abordagens baseadas em EEG, especificamente, alfa (α), beta (β), delta (δ), theta (θ), e a abordagem CogEff estão correlacionadas com o tempo decorrido para resolver tarefas de compreensão de código. Em particular, a abordagem CogEff possui uma correlação positiva de 0,3742 em relação ao tempo gasto para resolver tarefas de compreensão de código. Isto significa que, enquanto o valor da CogEff aumenta, o tempo para resolver uma tarefa de compreensão também aumenta. Além disso, essa correlação com o tempo decorrido foi significativa, onde *p-value* < 0,0001. Este é um comportamento coerente com a imposição de carga cognitiva nos desenvolvedores de software pois, é esperado que uma tarefa exija mais tempo para resolvê-la após uma crescente imposição de carga cognitiva aos desenvolvedores.

As abordagens alfa (α), theta (θ), beta (β), Delta (δ) apresentaram correlação negativa em relação ao tempo para resolver as tarefas de compreensão de código. Isso é coerente com o comportamento oscilatório das ondas cerebrais em relação ao grau de imposição de carga cognitiva nas tarefas. Por exemplo, as ondas cerebrais tendem a dessincronizar durante tarefas que demandam esforço mental, em relação a períodos de descanso (KLIMESCH, 1999).

Os resultados da Tabela 32 apresentam correlações na mesma direção para os indicadores em relação aos resultados da Tabela 31. Mesmo que as correlações no geral, foram mais fracas, os *p-values* são aproximados do intervalo de significância, isto é, de 0,05. Desta forma, ainda assim, os resultados são coerentes com as correlações apresentados na Tabela 32.

A Figura 20 ilustra a correlação geral a correlação de *Pearson* das abordagens EEG e o tempo decorrido para completar as tarefas de compreensão. A abordagem CogEff (A) apresenta uma linha horizontal que inclina da esquerda para direita de modo crescente. Essa correlação representa um aumento nos valores da cogEff assim que o tempo também aumenta. Enquanto as abordagens alpha (B), theta (C), beta (D) e delta (E), apresentaram nesses testes o comportamento inverso, ou seja, ao passo que o tempo para concluir a tarefa diminui, o valor desses indicadores aumenta. Além disso, a linha de correlação do *Event-Related Desynchronization*

Tabela 31: Resultado dos testes da Hipótese 2 com correlação de *Pearson*.

Tarefas	Estatísticas	Abordagens EEG							Tempo
		Alfa	Theta	Beta	Delta	ERD	ASR	CogEff	Médio (s)
1	**Correlação**	**-0,7239**	**-0,6908**	**-0,8113**	**-0,6373**	0,2514	-0,2975	**0,4951**	24,3
	p-value	**<0,0001**	**<0,0001**	**<0,0001**	**0.00003**	0,1451	0,0826	**0,0024**	
2	**Correlação**	**-0,5847**	**-0,5875**	**-0,5572**	**-0,6602**	-0,1127	0,1304	0,2919	40,9
	p-value	**0,0002**	**0,0002**	**0,0005**	**0,00001**	0,5191	0,455	0,0888	
3	**Correlação**	**-0,5901**	**-0,5901**	**-0,5900**	**-0,5899**	**-0,3712**	0,0662	**0,3748**	27,9
	p-value	**0,0001**	**0,0001**	**0,0001**	**0,0001**	**0,0281**	0,7052	**0,0264**	
4	**Correlação**	**-0,7156**	**-0,8236**	**-0,6266**	**-0,3395**	-0,1559	-0,0107	0,1593	36,6
	p-value	**<0,0001**	**<0,0001**	**0,00005**	**0,0459**	0,371	0,9513	0,3607	
5	**Correlação**	**-0,6919**	**-0,6550**	**-0,8235**	**-0,5944**	-0,1813	-0,0057	**0,6991**	35,1
	p-value	**0,000004**	**0,00001**	**<0,0001**	**0,0001**	0,2971	0,9741	**<0.0001**	
6	**Correlação**	**-0,5899**	**-0,5897**	**-0,5898**	**-0,5896**	**-0,3785**	0,0995	**0,5269**	51,8
	p-value	**0,0001**	**0,0001**	**0,0001**	**0,0001**	**0,0249**	0,5692	**0,0011**	
7	**Correlação**	**-0,5893**	**-0,5894**	**-0,5893**	**-0,5897**	0,2741	**-0,0360**	**0,5001**	44,5
	p-value	**0,0001**	**0,0001**	**0,0001**	**0,0001**	0,111	**0,8371**	**0,0022**	
8	**Correlação**	**-0,1120**	-0,1570	**-0,2328**	-0,5388	0,0007	-0,2104	0,0741	46,1
	p-value	**0,5218**	0,3676	**0,1783**	0,0008	0,9966	0,2249	0,6721	
9	**Correlação**	**-0,5819**	**-0,5831**	**-0,5821**	**-0,5978**	**-0,1428**	0,0907	0,1022	48,2
	p-value	**0,0002**	**0,0002**	**0,0002**	**0,0001**	**0,4131**	0,6041	0,559	
10	**Correlação**	-0,5910	-0,5900	-0,5923	-0,8096	0,2211	0,2265	0,4899	48,7
	p-value	<0,0001	<0,0001	<0,0001	<0,0001	0,2017	0,1906	0,0028	
Todas	**Correlação**	**-0,3410**	**-0,3862**	**-0,3935**	-0,4902	-0,0578	0,0009	**0,3742**	50,9
	p-value	**<0,0001**	**<0,0001**	**<0,0001**	**<0,0001**	0,2803	0,9861	**<0,0001**	

Fonte: Elaborado pelo autor.

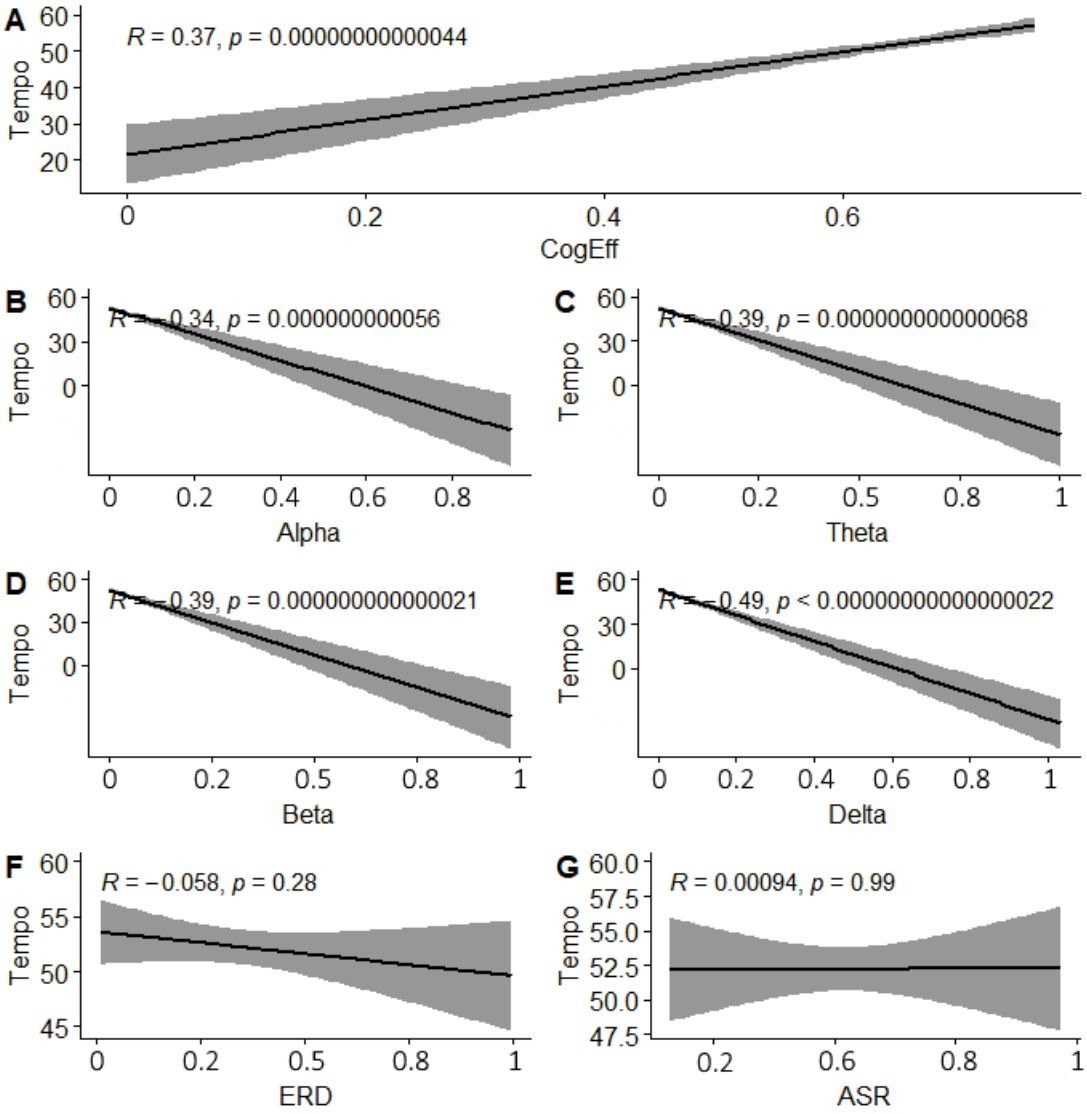
Tabela 32: Resultado dos testes da Hipótese 2 com correlação de *Spearman*.

Tarefas	Estatísticas	Abordagens EEG							Tempo
		Alfa	Theta	Beta	Delta	ERD	ASR	CogEff	Médio (s)
1	**Correlação**	**-0,3539**	-0,3539	**-0,3292**	-0,3052	0,1084	-0,0558	0.0774	24,3
	p-value	**0,0369**	0,0369	**0,0534**	0,0745	0,535	0,7499	0.6582	
2	**Correlação**	-0,2490	**-0,3507**	-0,2626	**-0,2927**	0,0216	0,0891	-0,0788	40,9
	p-value	0,1492	**0,03884**	0,1275	**0,08784**	0,9019	0,6106	0,6524	
3	**Correlação**	-0,2694	-0,2522	-0,2668	-0,1109	**-0,5522**	0,0846	0,1825	27,9
	p-value	0,1176	0,1437	0,1212	0,5257	**0,0005**	0,6286	0,2938	
4	**Correlação**	**-0,3638**	**-0,3520**	-0,3313	-0,1907	0,1230	0,1239	0,1429	36,6
	p-value	**0,0316**	**0,03805**	0,0518	0,2725	0,4812	0,4782	0,4125	
5	**Correlação**	**-0,2715**	-0,3222	**-0,2927**	-0,2387	-0,2047	0,1341	0,0825	35,1
	p-value	**0,1145**	0,05902	**0,0878**	0,1672	0,238	0,4422	0,6374	
6	**Correlação**	-0,2365	-0,2353	-0,2772	-0,0842	**-0,1459**	0,1308	0,1265	51,8
	p-value	0,1713	0,1734	0,1069	0,6303	**0,4029**	0,4535	0,4688	
7	**Correlação**	**-0,3152**	**-0,3099**	**-0,3563**	-0,3099	0,3110	**0,0167**	0,0547	44,5
	p-value	**0,0650**	**0,07002**	**0,03561**	0,0700	0,0689	**0,9241**	0,7548	
8	**Correlação**	-0,1477	-0,2621	-0,1619	-0,2669	0,1003	0,0051	-0,0175	46,1
	p-value	0,3969	0,1281	0,3526	0,1211	0,5663	0,9764	0,9203	
9	**Correlação**	-0,1061	-0,0615	-0,0948	-0,1787	-0,2230	0,1909	0,2763	48,2
	p-value	0,5437	0,7254	0,5879	0,3042	0,1978	0,2718	0,1081	
10	**Correlação**	-0,3272	**-0,3654**	-0,3320	**-0,3459**	0,1221	0,2924	0,2188	48,7
	p-value	0,0549	**0,0308**	0,0513	**0,0417**	0,4844	0,0882	0,2066	
Todas	**Correlação**	**-0,2695**	**-0,2902**	**-0,2718**	**-0,2267**	-0,0434	0,0983	**0,1166**	50,9
	p-value	**<0,0001**	**<0,0001**	**<0,0001**	**<0,0001**	0,4178	0,0661	**0.0291**	

Fonte: Elaborado pelo autor.

(F) e *Asymmetry Ratio* (G) estão praticamente horizontais e, portanto, evidenciando que, neste cenário, que não há correlação com o tempo decorrido para resolver tarefas de compreensão.

Figura 20: Representação da correlação de *Pearson* entre as abordagens EEG e o tempo para concluir as tarefas de compreensão.



Fonte: Elaborado pelo autor.

Os resultados em geral mostram que a CogEff reflete um comportamento coerente com a carga cognitiva. Os resultados da CogEff aumentam enquanto o tempo aplicado para resolver as tarefas de compreensão de código também aumentam. A vantagem disso é apresentar um valor intuitivo a carga cognitiva imposta aos desenvolvedores de software, diferentemente dos valores brutos das demais abordagens utilizadas para mensurar a carga cognitiva na engenharia de software. Além disso, a CogEff poderia ser usada para complementar as outras abordagens já que considera regiões relacionadas a compreensão de código.

Conclusão QP2: *os resultados de correlação de Pearson e Spearman mostram que a CogEff possui correlação positiva em relação ao tempo para realizar tarefas de compreensão contidas no conjunto de dados utilizado. Esse resultado indica que, enquanto o valor da CogEff aumenta o tempo para completar tarefas de compreensão também aumenta. Esse resultado também significa que, ao passo que o valor da CogEff diminui, o tempo para completar tarefas de compreensão também diminui. Esse resultado indica que a CogEff produz um valor coerente com a carga cognitiva dos desenvolvedores. Por fim, essa correlação também foi significativa, rejeitando a hipótese nula H_{2-0} e confirmando a hipótese alternativa H_{2-1}.*

5.8.4 QP3: Resultados da Hipótese 3 - diferença entre os valores da CogEff entre tarefas de complexidade menor e maior

Esta questão investiga a questão de pesquisa 3 sobre a avaliação da abordagem CogEff. Especificamente, foi analisado se existe diferença no indicador CogEff em tarefas de complexidade menor (*complMenor*) em relação a tarefas de complexidade maior (*complMaior*). A Tabela 33 e a Tabela 34 apresentam os resultados dos testes de Wilcoxon e *t-test* respectivamente. Os resultados dessas tabelas são descritos a seguir.

Teste de Wilcoxon. A Tabela 33 apresenta o resultado do teste de Wilcoxon entre os valores da CogEff entre os grupos de tarefas com complexidade menor (*complMenor*), e complexidade maior (*complMaior*). O resultado do teste de Wilcoxon apresenta que existe uma diferença significativa no indicador da CogEff entre tarefas com complexidade maior (*complMenor*) e tarefas com complexidade menor (*complMenor*). Especificamente, um teste não pareado de Wilcoxon apresentou que o valor da CogEff no grupo de tarefas de complexidade maior (*complMaior*) apresentou um aumento de maneira estatisticamente significativa em relação às tarefas de compreensão com complexidade menor (*complMenor*), especificamente com $p - value < 0,0001$. A média e a mediana da CogEff aumentaram no grupo de tarefas de complexidade maior (*complMaior*).

Tabela 33: Resultado da Hipótese 3 com o teste de Wilcoxon

	N	Mediana	Média	Média de Ranks	Mediana das diferenças	Wilcoxon rank	Intervalo de confiança		P-valor
							Inferior	Superior	
Complexidade Menor	778	0,72	0,72	0,72	-0,02	173491	-0,025	-0,018	< 0,0001
Complexidade Maior	614	0,75	0,75	0,75					

Fonte: Elaborado pelo autor.

T-test. O *t-test* confirma o resultado obtido no teste de Wilcoxon. Dessa forma, a diferença do indicador CogEff é estatisticamente significativa entre o grupo de tarefas de compreensão de complexidade menor (*complMenor*), e tarefas de compreensão de complexidade maior (*complMaior*). Especificamente, a Tabela 34 apresenta que t(-2,8) = 1338,4, p $<$ 0,005. Por causa das

médias da CogEff apresentada entre tarefas de complexidade diferentes, e a direção do valor de t, pode-se concluir que existe um aumento no valor da CogEff estatisticamente significativo nas tarefas de complexidade maior (*complMaior*) de 0,72 $\pm$ 0,17 a 0,75 $\pm$ 0,17 0,16, com $p - value < 0,005$. Especificamente, um aumento de 0,03 $\pm$ 0,02 no valor da CogEff em tarefas de complexidade maior (*complMaior*).

Tabela 34: Resultado da Hipótese 3 com o *t-test*.

	N	**t**	**df**	**Média**	**Desvio Padrão**	**Intervalo de confiança**		**P-valor**
						Inferior	Superior	
Complexidade Menor	778	-2,8	1338,4	0,72	0,17	-0,0433	0,0075	0,0052
Complexidade Maior	614			0,75	0,16			

Fonte: Elaborado pelo autor.

Conclusão QP3: *os resultados dos testes de Wilcoxon e t-test apresentaram que o valor da CogEff aumentou durante as tarefas de complexidade maior (complMaior). Este resultado pode indicar que tarefas de complexidade maior exigem mais carga cognitiva de desenvolvedores de software. Ambos os testes apresentaram uma diferença significativa entre os grupos de tarefas de complexidade menor (complMenor), e maior (complMaior), rejeitando a hipótese nula H_{3-0} e confirmando a hipótese alternativa H_{3-1}.*

5.9 Ameaças a validade

Foram mitigados vários aspectos para tratar ameças a validade em relação à avaliação da CogEff. Especificamente, aspectos internos, construção, conclusão e externos (WOHLIN et al., 2012) foram analisados.

Validação interna. As medidas de validação interna do estudo mitigam riscos que podem ameaçar a relação causal entre variáveis independentes e dependentes. O sinal de EEG contém vários componentes que podem ameaçar a relação entre atividade cerebral captadas pelas ondas elétricas e as variáveis independentes. Por isso, o primeiro passo da abordagem CogEff é atenuar vários componentes que não são relacionados a atividade neural de interesse para a avaliação da carga cognitiva tais como, os movimentos oculares, e os ruídos causados por dispositivos eletrônicos. Apesar disso, os sinais de EEG ainda podem conter ruídos e interferências.

Validação de construção. As medidas de validação de construção do estudo mitigam riscos que envolvem a generalização do estudo em relação as definições envolvidas tanto na abordagem quanto o experimento. Isto é, se o que foi medido está conforme o que foi definido. Para tratar esse risco os construtores da avaliação foram definidos, e detalhados. As propriedades e parametrizações da CogEff foram especificadas. Para evitar viés de métodos estatísticos, foram utilizados múltiplos testes estatísticos em cada questão de pesquisa. A questão de pesquisa 1 deste capítulo, por exemplo, utilizou os testes de *Spearman* e *Kendall* por serem apropriadas a testarem a relação entre uma variável contínua, isto é, os resultados das abordagens EEG, e

dados categóricos, que são as respostas. A questão de pesquisa 2 utilizou testes de *Pearson*, e *Spearman* pois essa questão testou a correlação entre duas variáveis contínuas, i.e., abordagens para mensurar carga cognitiva baseada no EEG com o tempo. A questão de pesquisa 3 utilizou dois testes estatísticos, isto é, de Wilcoxon e o *t-test* para aumentar a confiabilidade dos resultados. Além disso, os dados do *dataset* (SEGALOTTO, 2018) são constituídos por tarefas de compreensão que pertencem a cinco grupos de objetivos diferentes, as quais possuem uma versão menor e outra maior. Essas tarefas foram conduzidas de forma randomizada para não afetar versões menores das tarefas de compreensão não impactasse no aprendizado aos desenvolvedores ao realizarem as tarefas de compreensão com complexidade maior.

Validade externa. Referem-se as medidas adotadas para mitigar riscos em relação a generalização dos resultados em ambientes externos. Generalizar os resultados dessa avaliação é um desafio, uma vez que dependem de dados de EEG que são afetados por fatores humanos. Dessa forma, os dados contidos no *dataset* (SEGALOTTO, 2018) manteve uma homogeneidade dos perfis dos participantes. Por exemplo, a maioria amplamente passou pela mesma instituição de ensino e possuem entre dois e sete anos de experiência em desenvolvimento de software. Esses resultados se referem a um estudo realizado em laboratório, que fornece tarefas de compreensão de código. Nestas tarefas, participantes precisam realizar uma leitura e compreender um trecho de código, e deduzir o resultado desse trecho de código. Essas tarefas compreendem em 10 trechos de código em Java com limite de 60 segundos para serem concluídas.

Validade das conclusões. São as medidas tomadas para garantir que as conclusões deste estudo não estejam ameaçadas devidos a vieses contidos nas análises. Para reduzir esses riscos, um intervalo de confiança de 95% foi definido. Os testes estatísticos foram aplicados após seguirem os requisitos exigidos pelos testes utilizados. Eles são adequados em relação as variáveis que foram analisadas. Por fim, para evitar o *fishing-error*, as conclusões só foram definidas após a obtenção dos resultados, respeitando a relação causal entre as variáveis independentes e dependentes utilizadas.

5.10 Considerações finais

Esta seção avaliou a CogEff e as demais abordagens baseadas em EEG. Os resultados em geral mostram possíveis evidências de que a CogEff tem relação com tarefas de compreensão de código. Primeiro, a CogEff, através de ambos os testes de *Kendall* e *Spearman* apresentou ter correlação negativa com as respostas. Segundo motivo, é que a CogEff apresentou correlação positiva em relação ao tempo para concluir tarefas de compreensão. O resultado da correlação das principais abordagens EEG utilizados para mensurar a carga cognitiva reforçaram a relação da CogEff com compreensão de código, por apresentarem um comportamento semelhante. Por fim, através do valor obtido pela abordagem CogEff foi possível evidenciar uma diferença significativa entre um grupo de tarefas de compreensão com complexidade menor e outro de complexidade maior. Isso pode indicar que tarefas com complexidade mais altas impõe mais

carga cognitiva.

Para trabalhos futuros, pretende-se realizar a mesma avaliação, porém com parametrizações diferentes da CogEff. O próximo capítulo retoma a relação das abordagens baseadas com EEG com a compreensão de código empregando-as em técnicas de *machine learning*. Especificamente, as técnicas de *machine learning* classificam as respostas das tarefas de compreensão baseado nos dados de EEG de desenvolvedores. No próximo capítulo foi avaliado como a abordagem CogEff contribui com a efetividade das técnicas de *machine learning* para classificar compreensão de código.

6 CLASSIFICAÇÃO DE COMPREENSÃO DE CÓDIGO BASEADA EM DADOS DE EEG

Este capítulo busca responder à questão de pesquisa 4 desta tese: "Quão efetivas são as técnicas de *machine learning* para classificar compreensão de código usando dados de EEG?". Nos últimos anos, a academia e a indústria utilizaram abordagens relacionadas a carga cognitiva dos desenvolvedores aplicada a várias atividades relevantes no contexto da engenharia de software (MÜLLER; FRITZ, 2016; KOSTI et al., 2018; PEITEK et al., 2018a; MEYER; ZIMMERMANN; FRITZ, 2017; FRITZ et al., 2014). No entanto, estudos empíricos não se concentraram em avaliar o uso de técnicas de aprendizado de máquina para classificar a compreensão dos desenvolvedores utilizando dados de EEG. Ao invés disso, os dados de EEG foram usados em conjunto com técnicas de *machine learning*, como *Naïve Bayes* e *Support Vector Machine* para classificar problemas de qualidade de software (MÜLLER; FRITZ, 2016), indicar o nível de dificuldade de tarefas de programação (KOSTI et al., 2018; PEITEK et al., 2018a; MEYER; ZIMMERMANN; FRITZ, 2017; FRITZ et al., 2014), para classificar o nível de experiência dos desenvolvedores (CRK; KLUTHE; STEFIK, 2015; CRK; KLUTHE, 2016), e o nível da experiência do desenvolvedor (LEE et al., 2017) utilizando *Support Vector Machine*. Outro problema é que estudos (LEE et al., 2017; FRITZ et al., 2014; FUCCI et al., 2019) que treinaram técnicas de *machine learning* com dados de EEG em engenharia de software raramente verificaram se a eficácia das abordagens de aprendizado de máquina era maior do que um classificador que realiza classificações aleatórias. A ausência dessa avaliação limita as evidências da eficácia dos classificadores de *machine learning* em relação à classificadores aleatórios.

Portanto, um estudo empírico foi conduzido para avaliar a eficácia das técnicas de *machine learning* para classificar a compreensão do código dos desenvolvedores com base em dados de EEG, especificamente, a partir do *dataset* (SEGALOTTO, 2018), o qual foi utilizado no capítulo 5. Deste modo, a contribuição deste capítulo não se refere a coleta dos dados, mas sim ao propósito de utilização dos dados, ao tratamento dos dados, processamento dos dados, treinamento de classificadores de *machine learning*, variáveis, métodos de análise e avaliação dos resultados, resultados, e demais implicações. A compreensão de código se refere ao processo mental em que os desenvolvedores interpretam o código-fonte com o intuito de deduzir a saída do código-fonte. Este trabalho investiga a efetividade de técnicas de *machine learning* que pesquisadores geralmente utilizam para treinar com dados de EEG em aplicações na área de engenharia de software (FUCCI et al., 2019; LEE et al., 2017; MÜLLER; FRITZ, 2016). Além disso, os pesquisadores treinaram amplamente essas técnicas com dados de EEG em aplicativos BCI (LOTTE et al., 2018, 2007). Em particular, técnicas de *machine learning* foram treinadas com os dados de EEG dos desenvolvedores, incluindo a *K-Nearest Neighbor* (KNN), *Neural Network* (NN), *Naïve Bayes* (NB), *Random Forest* (RF) e *Support Vector Machine* (SVM). Essas técnicas de *machine learning* possuem diferentes características, que consequentemente podem impactar nos resultados ao serem treinados com dados de EEG. Por isso, essas diferenças

também são discutidas neste capítulo. Os classificadores são avaliados por meio da validação cruzada com 10 *Folds*. Além disso, a efetividade das das técnicas de *machine learning* foi realizada apenas com baseada na utilização de dados de EEG, e não considera nenhum tipo de dado externo, tais como a complexidade do código, ou o nível de experiência dos participantes. Embora o conjunto de dados disponibilize dados do perfil dos participantes, o foco deste capítulo é avaliar a eficácia na utilização de dados de EEG para classificar de compreensão de código. Desta forma, o foco deste capítulo é destacar exclusivamente o potencial desse tipo de dado para classificar compreensão de código. A efetividade das técnicas de aprendizado de máquina foi medida em relação a precisão, *recall*, e *f-measure*. Foi utilizado um *t-test* de amostra única para verificar se as *f-measures* são significativamente acima de 80%, e para testar se são significativamente maiores que a média de um classificador genérico que realiza classificações aleatórias.

6.1 Metodologia

Esta seção descreve os procedimentos e aspectos relativos à metodologia deste trabalho (GONÇALES et al., 2020). A Seção 6.1.1 define os objetivos e a questão de pesquisa deste capítulo. A Seção 6.1.2 descreve o processo utilizado para cumprir os objetivos deste capítulo. A Seção 6.1.3 apresenta as métricas utilizadas para medir a eficácia das técnicas de aprendizado de máquina analisadas. A Seção 6.1.4 apresenta os procedimentos de análise dos resultados.

6.1.1 Objetivo e questões de pesquisa

Utilizando o modelo *Goal, Question, Metric*, também conhecido como GQM (CALDIERA; ROMBACH, 1994), o objetivo deste capítulo está definido abaixo:

Analisar técnicas de *machine learning*
com o propósito de investigar seus efeitos
com respeito as suas efetividades
a partir do ponto de vista dos pesquisadores
no contexto de classificar a compreensão de código a partir de dados EEG.

Especificamente, este estudo busca avaliar a efetividade das técnicas de aprendizagem de máquina na classificação de compreensão de código a partir de dados EEG. A partir desse objetivo, foram formuladas três questões de pesquisa:

- **QP1:** quão efetivos são as técnicas de *machine learning* para classificar compreensão de código utilizando dados EEG?
- **QP2:** quais técnicas de *machine learning* alcançam uma efetividade maior que um classificador aleatório para classificar compreensão de código utilizando dados EEG dos desenvolvedores de software?

- **QP3:** qual é a efetividade das técnicas de *machine learning* para classificar compreensão de código caso os dados da CogEff sejam adicionados ao treinamento?

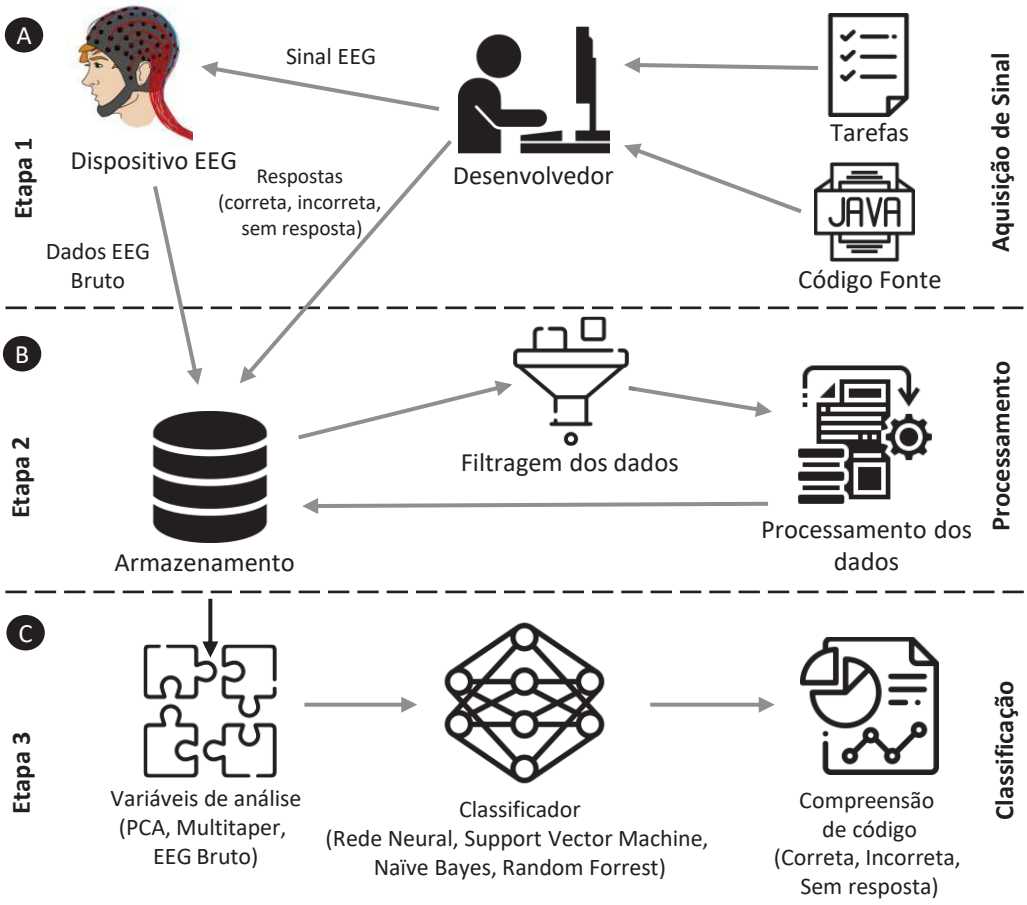
6.1.2 Método para coleta e processamento de dados

A Figura 21 apresenta o fluxo de processamento dos dados das ondas cerebrais de desenvolvedores de software. Este processo consistiu em três etapas: aquisição de sinal, processamento de sinal, e classificação de dados. A etapa de aquisição de sinal é a mesma utilizada em (SEGALOTTO, 2018), pois foi assim que os dados foram organizados. Os procedimentos, parâmetros, e implementações das etapas 2 e 3 são exclusivas desta tese. Estas etapas são descritas abaixo:

Etapa 1: Aquisição de sinal. A Figura 21.A apresenta como o sinal do EEG é coletado de desenvolvedores de software. O perfil dos desenvolvedores de software deste conjunto de dados encontra-se descrito no capítulo 5.3. O sinal é coletado com um EEG portátil, ou seja, um Emotiv Epoc+ (Emotiv Systems, 2021). Este dispositivo EEG tem 14 canais e lê os sinais emitido pela atividade cerebral de desenvolvedores de software em uma taxa de amostragem de 256Hz, isto é, 256 linhas de dados por segundo. Este experimento foi executado com o auxílio do software *Paradigm* (Paradigm Software, 2019), o qual foi responsável por gerenciar a sincronização dos sinais cerebrais do EEG em relação a cada tarefa de compreensão. Praticamente, o software apresentava as tarefas de compreensão aos desenvolvedores de software, enquanto inseria os marcadores (identificadores das respostas e tarefas) nos dados coletados do EEG. Os participantes executaram 10 tarefas de compreensão as quais foram apresentadas em ordem aleatória. Em um limite de 60 segundos os participantes tinham que compreender o trecho de código que lhes foi apresentado e deduzir a saída do código-fonte. Com base nesta saída, um marcador é registrado nos dados do usuário EEG indicando qual a tarefa sendo executada, e indicando também se a resposta estava correta, incorreta ou se o usuário não respondeu (não respondeu a tarefa dentro no tempo limite).

Etapa 2: Processamento de sinal. A Figura 21.B apresenta a segunda etapa deste estudo. O propósito desta etapa é processar e tratar os sinais brutos de EEG. **Filtragem de dados.** Todos os dados EEG de cada usuário são filtrados. Esta etapa de filtragem consiste em tratar os sinais de EEG. As seguintes ações foram tomadas para filtrar os sinais EEG: (1) Remoção de sinais com leituras anormais: os sinais com potência anormal (acima de 8 μVolts) foram ignorados, seguindo as recomendações do fabricante do EEG; (2) Filtro *High* e *Low-pass*: Foi aplicado o filtro *Low* e *High-pass* para atenuar sinais com frequências menores e maiores que as frequências de corte (0,6 e 45 Hertz). Dados inválidos para o experimento foram descartados, tais como as tarefas às quais os desenvolvedores de software resolveram em menos de cinco segundos. **Processamento dos dados.** Os sinais de EEG foram normalizados entre -1 e 1. Os marcadores de resposta (correto, incorreto ou "sem resposta") foram replicados para outras linhas de EEG porque esses marcadores estavam presentes apenas no momento exato em que os desenvolvedores responderam a tarefa. Os arquivos EEG foram processados em intervalos

Figura 21: Visão geral do fluxo de processamento dos dados EEG.



Fonte: Elaborado pelo autor.

de 5 segundos. **Balanceamento dos dados.** O número de amostras em cada uma das classes do conjunto de dados (correto, incorreto e sem resposta) não eram iguais e desbalanceados, por isso, a amostragem das classes dominantes foram balanceadas no conjunto de dados com o objetivo de equilibrar a quantidade de observações entre as classes. Este método é chamado de *undersampling*. Após o balanceamento dos dados, a quantidade de observações ficou próxima. Especificamente, após o balanceamento de dados, as classes correta, incorreta e "sem resposta" resultaram em 897, 440 e 941 observações por classe. As classes foram balanceadas usando a biblioteca padrão *SciKit-Learn* do ecossistema de linguagem *Python*.

Etapa 3: Classificação dos dados. A Figura 21.C apresenta a terceira etapa do estudo. O propósito desta etapa é classificar a compreensão de código dos desenvolvedores utilizando os dados pré-processados produzidos na etapa anterior. Os intervalos de frequência dos sinais EEG foram decompostos utilizando o método *Welch*. Os sinais EEG foram decompostos na banda alfa (α), beta (β), delta (δ), gama (γ) e theta (θ). Em seguida, foi realizado a Análise de Componentes Principais (PCA) para reduzir a dimensionalidade dos dados. Essa etapa de processamento no sinal EEG reduziu o tamanho original dos dados de 10 Gb para aproximadamente 2 Mb. **Classificadores.** Os classificadores foram avaliados utilizando o método de

validação cruzada em 10 *folds* (BENGIO; GRANDVALET, 2004). Os dados foram utilizados para treinar cinco classificadores: *K-Nearest Neighbors* (KNN), *Neural Network* (NN), *Naïve Bayes* (NB), *Support Vector Machine* (SVM) e *Random Forest* (RF). Além disso, essa pesquisa também treinou os dados EEG em um classificador aleatório, o qual praticamente advinha as classes de dados de modo randômico. A literatura também define esse tipo de classificador como "*random guesser classifier*". **Busca em Grid.** Para selecionar a configuração ideal para a arquitetura de *Neural Network* (NN), foi utilizado uma busca em *grid*. Este algoritmo explora vários parâmetros relacionados a arquitetura de rede neural. Este algoritmo realizou uma busca por parâmetros criando várias redes neurais, baseado nas seguintes informações e características:

- Quantidade de neurônios: foi configurado para a rede neural de 2, 4, 8 e 16 neurônios para cada camada;
- Quantidade de camadas escondidas: o algoritmo testou 1, 2, 10 e 20 camadas ocultas;
- Quantidade de épocas: o algoritmo realizou treinamentos iterando com 400, 800 e 1600 épocas.

Os resultados do algoritmo de busca em *grid* evidenciaram que a arquitetura com quatro camadas, ou seja, duas camadas externas e duas camadas ocultas é adequada para treinar dados de EEG em uma rede neural. Além disso, as camadas externas referem-se a camadas correspondentes à entrada e saída da rede. A primeira camada oculta também deve ser composta por 16 neurônios, e a segunda camada oculta devem conter oito neurônios. O algoritmo também foi executado em 800 épocas para reproduzir os resultados contidos nessa pesquisa.

Classificação. As técnicas de *machine learning* realizaram a classificação de situações de compreensão de código de desenvolvedores de software. O material suplementar para reproduzir os resultados da classificação encontra-se em GONCALES et al. (2021).

6.1.3 Métricas de efetividade

Estudos em engenharia de software adotaram precisão, *recall*, e *f-measure* (THARWAT, 2020) para analisar a efetividade das técnicas de *machine learning*, incluindo aqueles treinados com dados de EEG (LEE et al., 2017; FUCCI et al., 2019).

Especificamente essas métricas são compostas por três variáveis: verdadeiro positivo (*tp*), i.e., a quantidade de classificações que foram corretamente identificada pelas técnicas de *machine learning*; verdadeiro negativo (*tn*) a quantidade de classificações que as técnicas apropriadamente trataram como incorretas; falso positivo (*fp*), i.e., a quantidade de classificações em que o classificador tratou de maneira equivocada como correto; e falso negativo (*fn*), i.e., as classificações nas quais as técnicas de maneira equivocada considerou como incorreto. Baseado nessas definições, foram avaliadas as seguintes métricas de efetividade:

Precisão: representa a proporção de classificações corretas que realmente são corretas, i.e., classificações corretas não considerando falso positivos.

$$Precisão = \frac{tp}{tp + fp} \tag{6.1}$$

***Recall*:** representa a proporção de classificações corretas não considerando a proporção classificações falso negativas.

$$Recall = \frac{tp}{tp + fn} \tag{6.2}$$

***F-measure*:** representa a média harmônica entre o *recall* e a precisão.

$$F - Measure = 2 \cdot \frac{Precisão \cdot Recall}{Precisão + Recall} \tag{6.3}$$

A Tabela 35 descreve a escala para descrever a efetividade que foi utilizada neste trabalho. Deste modo, intervalos específicos são atribuídos a aspectos qualitativos.

Tabela 35: Categoria das métricas de efetividade.

Categoria	**Intervalo**
Muito Alto	0,80 $\leq$ métrica $\leq$ 1,00
Alto	0,60 $\leq$ métrica $<$ 0,80
Moderado	0,40 $\leq$ métrica $<$ 0,60
Baixo	0,20 $\leq$ métrica $<$ 0,40
Muito Baixo	0,00 $\leq$ métrica $<$ 0,20

Fonte: Elaborado pelo autor.

Métricas de efetividade na QP1. Foram medidas a precisão, *recall*, e *f-measure* (THARWAT, 2020) para responder a QP1. Essas métricas foram analisadas por que na QP1 foi analisada a efetividade das técnicas de *machine learning*. Além disso, foi avaliado o coeficiente de correlação de Matthews (BOUGHORBEL; JARRAY; EL-ANBARI, 2017), conhecido como MCC, na QP1 porque o MCC considera todos os valores da matriz de confusão. Deste modo, medindo o MCC é possível evidenciar se a precisão, *recall*, e *f-measure* estão coerentes. Valores do MCC próximos a -1 significa que a classificação é oposta a classificação esperada, e valores próximos a +1 significam resultados que correspondem a classificação esperada. Valores de MCC perto de zero significa que a técnica realizou classificações aleatórias.

Métrica de efetividade na QP2. Foram medidas a média harmônica entre *recall* e precisão das técnicas de *machine learning* na QP2, i.e., a *f-measure*. A *f-measure* foi medida porque seus resultados tendem a penalizar efetividades que possuam apenas a precisão alta, mas o *recall* baixo.

6.1.4 Procedimentos de análise

Análise descritiva. Foram analisados a distribuição da precisão, *recall*, e *f-measure*. O teste de *Shapiro Wilk* e *Kolmogorov Smirnov* foram aplicados para verificar a normalidade dos dados. Estes testes apontam um p-valor > 0.05 para demonstrar uma distribuição normal.

QP 1. Na questão de pesquisa 1 deste capítulo, a efetividade das técnicas de *machine learning* é descrita e categorizada em relação à Tabela 35 da Seção 6.1.3. Essa análise foi feita com base na Tabela 35 com o objetivo de fornecer a magnitude da efetividade das técnicas de *machine learning* para classificar compreensão de código baseada em dados EEG. Foi utilizado um *t-test* de uma amostragem para avaliar se as técnicas possuem efetividade maior que 80%.

QP 2. Na questão de pesquisa 2 deste capítulo, o *t-test* de uma amostragem foi utilizado para testar se as técnicas de *machine learning* tem efetividade maior que um classificador que realiza classificações aleatórias. Deste modo, a análise na questão de pesquisa 2 teve a intenção de verificar se a média de *f-measure* dos respectivos classificadores são maiores que a média do *f-measure* de um classificador que apenas advinha as classificações.

QP 3. Na questão de pesquisa 3 deste capítulo, é utilizado o *t-test* entre a efetividade das técnicas de *machine learning* treinadas com abordagens EEG tradicionais e das técnicas treinadas com abordagens EEG tradicionais junto com a abordagem CogEff.

6.2 Resultados

Esta Seção apresenta os resultados das questões de pesquisa definidas neste capítulo. Seção 6.2.1 apresenta a estatística descritiva dos resultados. Seção 6.2.2 descreve os resultados da QP 1. Seção 6.2.3 apresenta os resultados da QP 2.

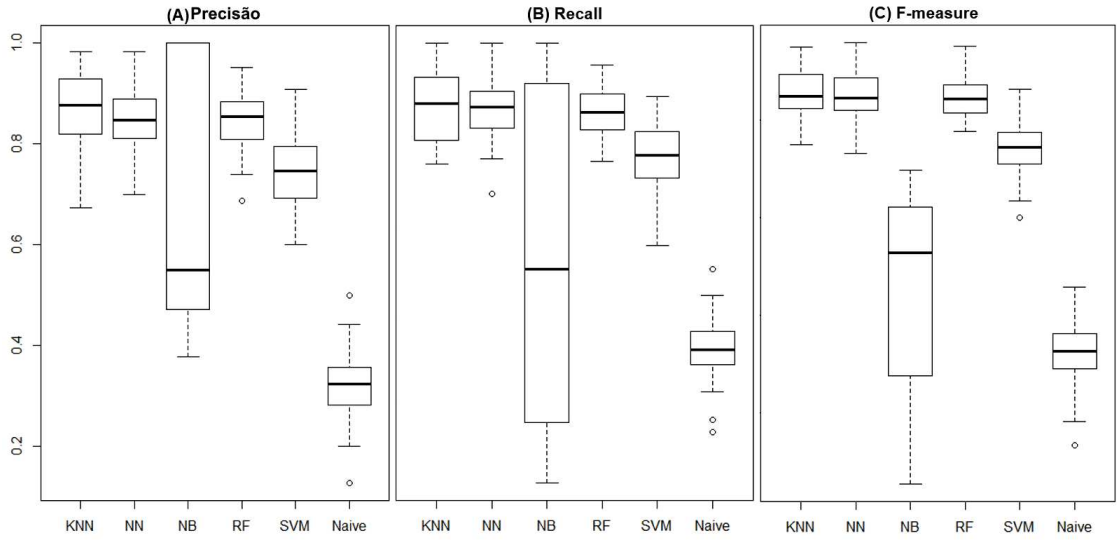
6.2.1 Análise descritiva

A Figura 22 apresenta a distribuição da efetividade de cada classificador em relação às métricas de (A) Precisão, (B) *Recall* e (C) *F-Measure*. Esses valores referem-se a precisão, *recall* e *f-measure* em relação a cada *fold*.

Os resultados mostram que as técnicas *K-Nearest Neighbor* (KNN), *Neural Network* (NN), *Random Forest* (RF) e *Support Vector Machine* (SVM) têm efetividade acima de 70% na precisão, *recall* e *f-measure*. Os resultados de efetividade de *Naïve Bayes* (NB) variou entre 20% a 80%, i.e., apresentou alta variação. O classificador *Naive*, o qual randomicamente advinha a classificação, obteve uma efetividade abaixo de 40% em todas as métricas.

Foram conduzidos testes de Shapiro-Wilk, e Kolmogorov Smirnov para verificar a normalidade dos dados. Especificamente, a efetividade dos classificadores *K-Nearest Neighbor*, *Neural Network*, *Random Forest*, *Support Vector Machine* e *Naïve Bayes* possuem distribuição normal em relação a precisão, *recall* e *f-measure* porque os testes de normalidade possuem um *p-value*

Figura 22: Distribuição da precisão, *recall*, e *f-measure* das técnicas de *machine learning*.



Fonte: Elaborado pelo autor.

$>$ 0,05. Os resultados de efetividade da técnica *Naïve Bayes* apresentou distribuição normal apenas na métrica *f-measure*, mas os testes evidenciaram que os resultados da precisão e *recall* não se enquadravam na distribuição normal.

6.2.2 QP1: Efetividade das técnicas de machine learning

A questão de pesquisa 1 investiga o quão efetivas são as técnicas de *machine learning* para classificar compreensão de código utilizando dados EEG. Este capítulo organiza os resultados de acordo com as categorias de métricas descritas na Seção 6.1.3. A Tabela 36 apresenta uma visão geral dos resultados em relação a precisão, *recall*, e *f-measure* das técnicas de *machine learning*. Nesta pesquisa, foram reportados a macro precisão, macro *recall*, e a macro *f-measure*. A versão macro desta métricas consiste na média da precisão, *recall*, e *f-measure* que cada classificador obteve por *fold*. Os *folds* são as partes do conjunto de treinamento através do método de validação cruzada. Os classificadores realizaram uma classificação multi-classe, i.e., o sinal EEG foi classificado de acordo com três classes de respostas: corretas, incorretas, e "sem resposta". Por exemplo, a macro precisão é igual a média entre os valores obtidos entre cada uma dessas classes. A mesma fórmula é aplicada para o macro *recall* e macro *f-measure*. Foi optado por reportar a versão macro por representar uma visão geral da efetividade dos classificadores por *fold*. Os resultados da precisão, *recall* e *f-measure* obtidos por classe são apresentados na apêndice C.

Precisão. Figura 23 apresenta os resultados da precisão das técnicas de *machine learning* obtido em cada *fold*. Na Tabela 36, a coluna precisão apresenta os resultados de cada classificador. **Muito Alto**. A maioria das técnicas de *machine learning* apresentou resultados de precisão muito alta ($\geq$ 0,80). As técnicas *K-Nearest Neighbor*, *Neural Network*, e *Random Forest* alcan-

Tabela 36: Resultado da precisão, *recall* e *f-measure* das respectivas técnicas de *machine learning*.

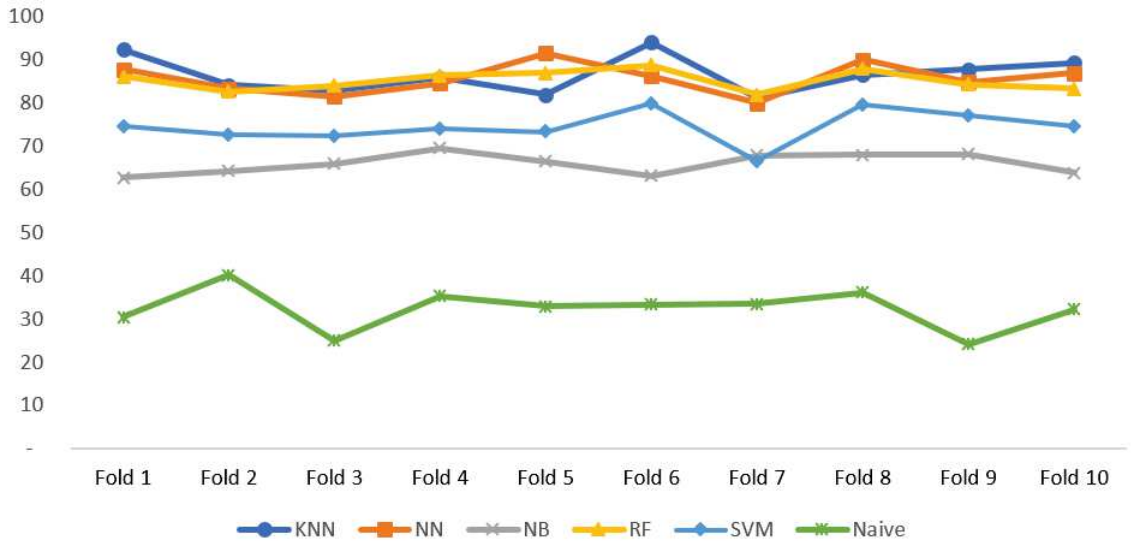
	KNN			NN			NB			RF			SVM			Naive		
	Prec.	Rec.	F1-m	Prec.	Rec.	F1-m	Prec.	Rec.	F1-m	Prec.	Rec.	F1-m	Prec.	Rec.	F1-m	Prec.	Rec.	F1-m
Fold 1	**92,4**	**91,6**	**91,7**	**87,8**	**87,2**	**87,4**	62,7	51,2	46,0	**86,1**	**85,0**	**85,0**	74,6	74,3	74,4	30,5	30,2	30,2
Fold 2	**84,2**	**84,4**	**84,0**	**83,4**	**83,7**	**83,5**	64,3	50,4	44,1	**82,6**	**83,0**	**82,7**	72,8	73,1	72,8	40,2	40,8	40,3
Fold 3	**82,9**	**82,2**	**82,3**	**81,6**	**81,7**	**81,6**	66,0	52,7	46,6	**84,0**	**83,6**	**83,7**	72,6	73,0	72,7	25,0	24,7	24,6
Fold 4	**85,8**	**84,9**	**84,2**	**84,6**	**84,6**	**84,2**	69,6	59,8	51,9	**86,4**	**86,0**	**85,5**	74,2	74,2	74,0	35,4	35,8	35,5
Fold 5	**82,0**	**81,9**	**82,0**	**91,5**	**91,2**	**91,3**	66,5	44,7	36,8	**87,0**	**86,3**	**86,6**	73,4	72,2	72,5	33,0	33,3	32,5
Fold 6	**94,0**	**93,0**	**93,2**	**86,3**	**85,0**	**85,2**	63,2	48,7	45,1	**88,9**	**88,7**	**88,7**	**80,0**	79,9	79,8	33,3	33,2	33,1
Fold 7	**81,2**	**81,5**	**81,0**	**80,0**	79,8	79,3	68,0	50,9	43,0	**82,0**	**82,0**	**81,8**	66,6	66,8	65,8	33,5	33,1	33,2
Fold 8	**86,5**	**87,0**	**86,3**	**90,1**	**89,5**	**88,8**	68,0	50,4	42,5	**88,0**	**87,9**	**87,9**	79,6	**80,4**	79,3	36,2	36,2	35,3
Fold 9	**87,9**	**87,0**	**87,1**	**84,9**	**85,3**	**85,1**	68,3	54,7	48,4	**84,4**	**84,7**	**84,3**	77,1	76,9	76,8	24,1	23,9	23,6
Fold 10	**89,4**	**88,4**	**88,5**	**87,1**	**86,3**	**86,5**	63,8	52,1	45,9	**83,4**	**83,3**	**83,3**	74,8	73,9	73,8	32,2	31,9	31,9
All	**86,6**	**86,2**	**86,0**	**85,7**	**85,4**	**85,3**	**66,0**	**51,6**	**45,0**	**85,3**	**85,1**	**85,0**	**74,6**	**74,5**	**74,2**	**32,4**	**32,3**	**32,0**

Legend:
KNN: *K-Nearest Neighbor*, NN: *Neural Network*, NB: *Naïve Bayes*, RF: *Random Forest*, SVM: *Support Vector Machine*,
Prec: Precisão, Rec.: *Recall*, F1-m: *F-Measure*

Fonte: Elaborado pelo autor.

çaram uma precisão muito alta para classificar a compreensão de código dos desenvolvedores. KNN alcançou uma precisão de 94% no *fold* 6, e a menor precisão alcançou 81,2% no *fold* 7. A técnica *Neural Network* alcançou 91,5% no *fold* 5, e a precisão mais baixa da técnica *Neural Network* foi no *fold* 7 com 80% de precisão. A menor precisão da *Random Forest* foi de 82% no *fold* 7, e a maior precisão foi de 88,9% no *fold* 6. KNN alcançou a maior precisão média entre todos os *folds* (86,6%), seguido pela NN (85,7%) e RF (85,3%). **Alta**. O classificador *Naïve Bayes* e *Support Vector machine* obtiveram alta precisão ($\leq$ 0,80). O classificador SVM alcançou uma precisão maior (74,6%) em relação a NB (66%). **Baixa**. O classificador *Naive* teve uma precisão baixa, i.e., uma média de 32,4%, e uma precisão máxima de 40% para classificar compreensão de código dos desenvolvedores.

Figura 23: Resultados da precisão das técnicas de *machine learning*.

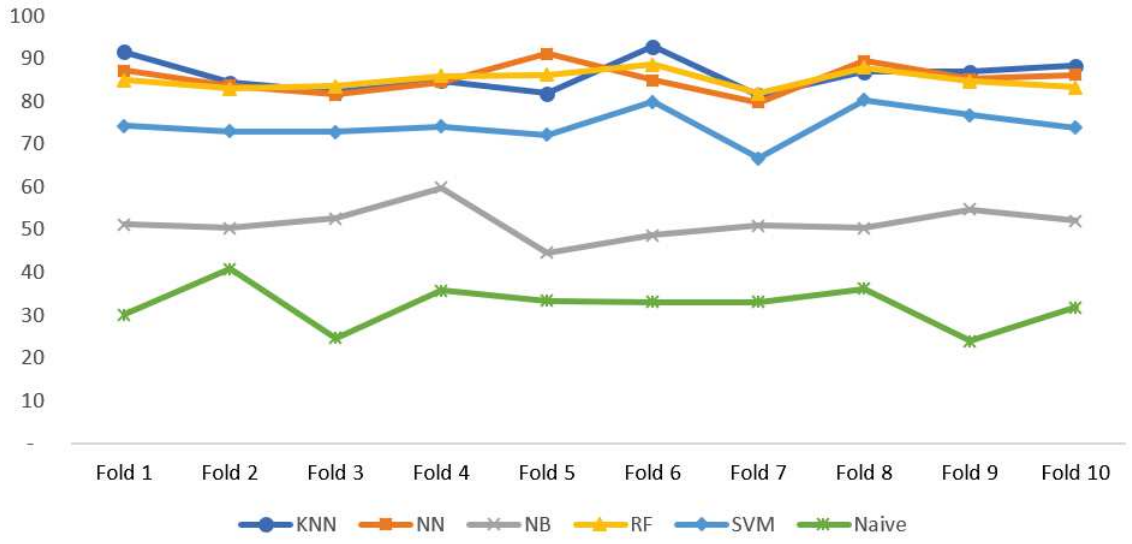


Fonte: Elaborado pelo autor.

Recall. A Figura 24 apresenta os resultados do *recall* das técnicas de *machine learning* obtidos em cada *fold*. Na Tabela 36, a coluna *recall* apresenta os resultados correspondentes de

cada classificador por *fold* na Figura 24. **Muito Alto**. KNN, NN, e RF apresentaram um *recall* médio muito alto para classificar compreensão de código dos desenvolvedores utilizando dados EEG. KNN apresentou a maior valor médio de *recall* (86,2%), seguido pelo NN (85,4%) e RF (85,1%). KNN obteve maior média de *recall* entre NN e RF porque os valores foram predominantemente altos na *fold* 1, *fold* 6, *fold* 9 e *fold* 10. **Alto**. O classificador SVM apresentou uma média de *recall* alto (74,5%). O valor de *recall* mais baixo ocorreu no *fold* 7 (66,8%), e o maior *recall* no *fold* 8 (80,4%). O classificador SVM manteve a efetividade do *recall* na mesma categoria que o SVM obteve na precisão. **Moderado.** NB apresentou uma média de *recall* moderada (51,6%). O *recall* da NB é menor que o valor da precisão, mas ainda é maior que o *recall* do classificador *Naïve*. **Baixo.** O classificador *Naive* apresentou uma média de *recall* baixa (32,3%), o que é coerente com uma técnica que apenas realiza classificações aleatórias.

Figura 24: Resultados do *recall* das técnicas de *machine learning*.

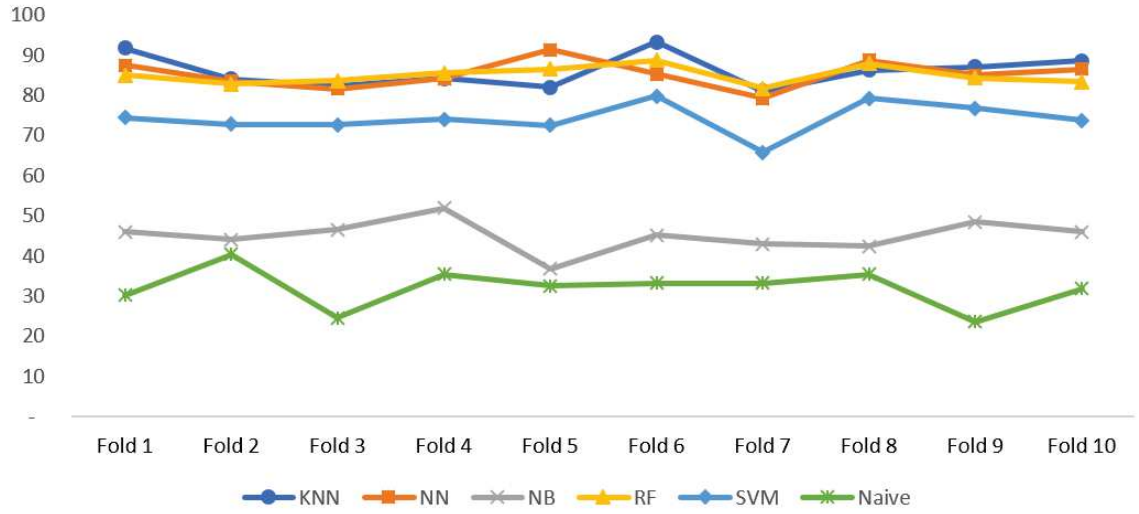


Fonte: Elaborado pelo autor.

F-measure. A Figura 25 apresenta o resultado da *f-measure* das técnicas de *machine learning* em cada *Fold*. Na Tabela 36, a coluna *f-measure* apresenta os resultados na Figura 25. *f-measure* é a média harmônica entre precisão e *recall*. Em geral, os resultados da *f-measure* são em geral mais baixos que os valores da precisão e do *recall*. A *f-measure* é uma métrica que penaliza as técnicas de *machine learning* que apresentam uma discrepância entre precisão e *recall*. **Muito Alto**. *K-Nearest Neighbors*, *Neural Network* e *Random Forest* apresentaram um *f-measure* muito alto para classificar compreensão de código dos desenvolvedores baseados em dados EEG. O maior valor médio da *f-measure* foi de 86% pela técnica KNN, seguido da NN (85,3%) e RF (85%). O menor valor da *f-measure* obtido por esses classificadores foi no *fold* 7, onde a técnica KNN obteve 81% de *f-measure*, *Neural Networks* alcançou 79,3% de *f-measure*, e RF alcançou no mínimo 81.8% de *f-measure*. O maior valor de *f-measure* da KNN alcançou 93,2% no *fold* 6, *Neural Networks* apresentou seu maior valor no *fold* 5 com 91,3% de *f-measure*, e a técnica *Random Forest* obteve 88,7% no *fold* 5. Um valor de *f-measure*

muito alto significa que os valores da precisão e o *recall* tem níveis altos, e possuem valores equilibrados. **Alto**. A técnica *Support Vector Machine* obteve um valor médio de *f-measure* alto (74,2%). Esta técnica permaneceu na mesma categoria da precisão e *recall*. **Baixo**. A técnica *Naïve Bayes* obteve um valor médio de *f-measure* baixo (45%) para classificar compreensão de código dos desenvolvedores baseados em dados EEG. A categoria de efetividade da NB é menor com a *f-measure*. A técnica NB obteve um *recall* moderado e uma precisão alta. De acordo com os valores da *f-measure*, nota-se que NB possui uma efetividade próxima ao classificador *Naive*. A técnica NB e classificador *Naive* possuem uma efetividade mais próxima nos *folds* 2, 5 e 8.

Figura 25: Resultados da *f-measure* das técnicas de *machine learning*.

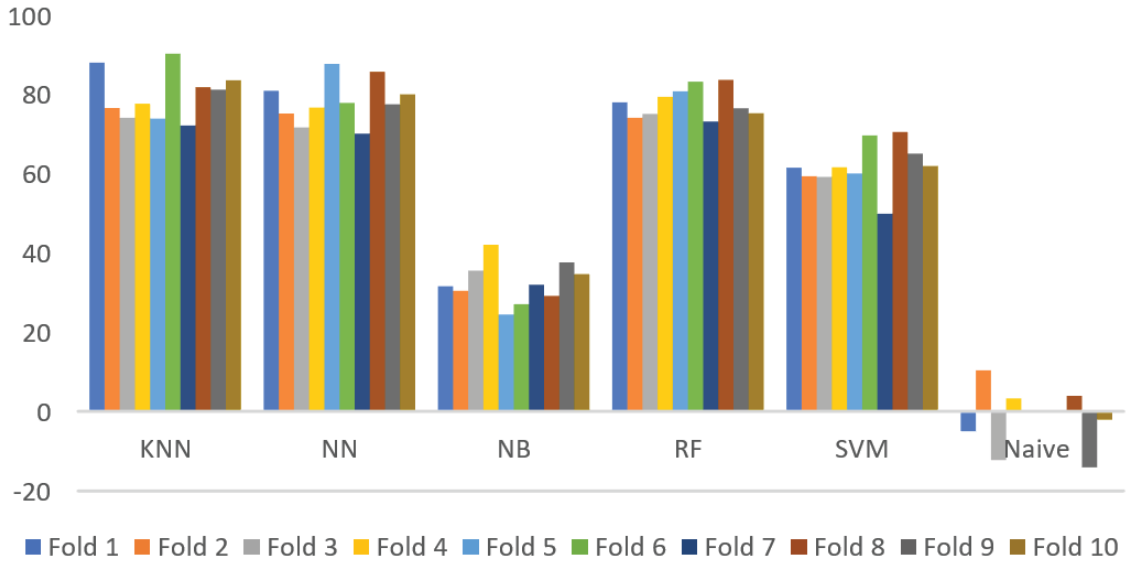


Fonte: Elaborado pelo autor.

Mesmo que a *f-measure* seja aplicada para medir efetividade das técnicas de *machine learning*, a *f-measure* não considera todos os valores da matriz de confusão. A falta desses valores pode gerar uma interpretação enviesada. Desta forma, além das métricas discutidas anteriormente, também foi calculado o MCC para medir a efetividade das técnicas de *machine learning* para classificar compreensão de código baseados em dados EEG.

MCC. A Figura 26 apresenta os resultados do MCC das técnicas de *machine learning*. Como mencionado anteriormente na Seção 6.1.3, a classificação está coerente com os resultados esperados quando os valores do MCC estão próximos a 1, mas quando os valores do MCC são próximos de -1 isto significa que as classificações das técnicas resultaram em valores diferentes dos esperados. MCC próximo a zero significa que as técnicas estão realizando classificações aleatórias. **Muito Alto**. KNN obteve uma média do de MCC de 0,8, o que indica que esta técnica realizou classificações precisas. **Alta**. Na perspectiva do MCC, a *Neural Network* e *Random Forest* apresentaram a mesma média (0,76). Este valor do MCC indica que esses classificadores foram efetivos para classificar a compreensão de código dos desenvolvedores baseado em dados EEG. **Baixo**. O classificador *Naïve Bayes* apresentou uma média de MCC de

Figura 26: Resultados do MCC das técncias de *machine learning*.



Fonte: Elaborado pelo autor.

0,32 indicando que a efetividade para classificar a compreensão de código dos desenvolvedores baseados em dados EEG pela técnica *Naïve Bayes* foi baixa. O valor do MCC apresenta que ao menos o classificador *Naïve Bayes* realiza classificações com um desempenho um pouco a cima de classificadores aleatórios. **Muito Baixo**. Como esperado, o classificador *Naive* obteve uma média MCC de -0.2, o que confirma que realmente a técnica está gerando classificações randômicas.

Efetividade acima de 80%. Também foi investigado nessa questão de pesquisa quais técnicas de *machine learning* apresenta uma efetividade significativamente $> 80\%$ para classificar a compreensão de código dos desenvolvedores, i.e., se os valores de *f-measure* estatisticamente pertencem ao grupo de alta efetividade definida na Tabela 35. Para esse teste, foi aplicado o *t-test* de uma amostragem para verificar se a média da *f-measure* das técnicas de *machine learning* está acima de 80%. Esta hipótese é verdadeira quando *p-value* $< 0{,}05$.

A Tabela 38 mostra que os respectivos *f-measure* dos classificadores *K-Nearest Neighbor*, *Neural Networks* e *Random Forest* possuem uma diferença média estatisticamente significante maior que 0,8 (*p-value* < 0.05). Porém, os classificadores *Naïve Bayes* e *Support Vector Machine* (SVM) falharam em rejeitar a hipótese nula e os respectivos *f-measure* desses classificadores possuem uma diferença média estatisticamente significante inferior a 0.8, deste modo falhando em fornecer alta efetividade para classificar a compreensão de código dos desenvolvedores baseado em dados EEG.

Observações finais. Os resultados da *f-measure* na Figura 25 apresenta que a técnica *Naïve Bayes* é menos efetiva para classificar compreensão de código dos desenvolvedores. Através da *f-measure* é possível constatar que a efetividade da *Naïve Bayes* é menor do que a sugerida pelas métricas de precisão na Figura 23. A métrica *f-measure*, ao contrário da precisão e do *recall*,

Tabela 37: Resultados do *t-test* de uma amostragem das *f-measures* dos classificadores, valor testado igual a 0.8.

	t	df	*p-value*	Dif. Média	95% IC	
					Inferior	Superior
K-Nearest Neighbors	6,187	29	<0,0001	0,060	0,0404	Inf
Neural Neworks	5,356	29	<0,0001	0,052	0,0327	Inf
Naïve Bayes	-9,935	29	1	-0,34	-0,4215	Inf
Random Forest	6,022	29	<0,0001	0,049	0,032	Inf
Support Vector Machine	-5,278	29	0,999	-0,058	-0,0807	Inf

Fonte: Elaborado pelo autor.

evidencia que a técnica *Naïve Bayes* possui uma efetividade muito próxima a um classificador aleatório. A *f-measure* evidencia uma penalidade a *Naïve Bayes* porque o resultado da *recall* é muito baixo. Na prática, a *f-measure* apresentou baixa efetividade porque a técnica de *Naïve Bayes* detectou classificações verdadeiras positivas (*tp*) com menos frequência comparada a outras técnicas de *machine learning*.

O *t-test* de uma amostragem (efetividade acima de 80%) evidenciou que a técnica *Support Vector Machine*, apesar de alcançar 80% em alguns *folds* na Tabela 36, não possui uma efetividade significativamente maior que 80%. Se baseando apenas na informação da Tabela 35, a técnica *Support Vector Machine* possui alta efetividade. O *t-test* e os valores da Tabela 36 confirmam que *K-Nearest Neighbor*, *Neural Network*, e *Random forest* possuem uma efetividade muito alta para classificar compreensão de código baseados em dados de EEG.

Conclusão QP1: *os classificadores K-Nearest Neighbors, Neural Network, e Random Forest foram efetivos ($\geq$ 80%) em relação à precisão, recall e f-measure para classificar a compreensão de código baseada em dados EEG. Esse resultado indica que os três classificadores possuem alta efetividade, conforme as categorias da Tabela 35, para classificar compreensão de código baseado em dados de EEG contidos na base de dados utilizada.*

6.2.3 QP2: Quais técnicas de aprendizagem de máquina alcançam uma efetividade maior que um classificador aleatório?

A questão de pesquisa 2 investiga quais das técnicas de *machine learning* tem uma média de *f-measure* que seja estatisticamente significante maior que um classificador *Naive*. A *f-measure* é a métrica de referência por considerar em seu cálculo ambas a precisão e o *recall*.

A Tabela 38 apresenta os resultados do teste de hipótese da efetividade de cada classificador em reação o valor médio da *f-measure* do classificador *Naive*. O valor testado representando o

classificador *Naive* foi o valor médio de *f-measure* de 0,33. A Tabela 38 apresenta o valor de t (t), o grau de liberdade (df), o *p-value*, a diferença média de (*f-measure* entre os respectivos classificadores, e os limites do intervalo de confiança (95%).

Tabela 38: Resultado do *t-test* de uma amostragem da *f-measure* dos classificadores.

	t	df	*p-value*	Dif Médias.	95% IC	
					Inferior	Superior
K-Nearest Neighbors	55,31	29	$<$ 0,0001	0,5401	0,520	Inf
Neural Neworks	53,9	29	$<$ 0,0001	0,5326	0,512	Inf
Naïve Bayes	3,70	29	0,0004	0,1301	0,058	Inf
Random Forest	64,16	29	$<$ 0,0001	0,5293	0,512	Inf
Support Vector Machine	38,24	29	$<$ 0,0001	0,4215	0,398	Inf

Fonte: Elaborado pelo autor.

Resultados do *t-test* na Tabela 38 apresentam que todas as técnicas de *machine learning* possuem respectivos *f-measure* com uma diferença média maior (*p-value* $<$ 0,05) que a *f-measure* do classificador *Naive*. A Tabela 38 também apresenta que o classificador KNN possui uma média com maior diferença em relação ao classificador *Naive*. Especificamente, a média da *f-measure* da técnica KNN (M = 0,84, SD = 0,05) foi maior que a *f-measure* média do classificador *Naive* de 0,33, uma diferença média estatisticamente significante de 0,54, 95% IC [0,52 a Inf], t(39) = 55,31, *p-value* $<$ 0.0001. Até mesmo o classificador NB obteve uma diferença média estatisticamente significante maior (*p-value* $<$ 0.05) em relação ao classificador *Naive*. Na questão de pesquisa anterior, o NB aparentava ter uma efetividade muito próxima ao classificador *Naive*. Este resultado indica que todos os classificadores possuem uma efetividade estatisticamente significante maiores do que um simples classificador aleatório (*Naive*). Deste modo, as técnicas estavam realmente classificando a compreensão de código dos desenvolvedores baseado nos dados EEG.

Observações finais. Até mesmo o classificador *Naïve Bayes* que obteve performance aproximada do classificador aleatório nas análises anteriores obteve uma efetividade superior em alguns *folds* em relação ao classificador *Naive*. Neste caso, a técnica *Naïve Bayes* também tem potencial de ser utilizada para classificar compreensão de código dos desenvolvedores. O teste demonstrou que o classificador *Naïve Bayes* não estava realizando a classificação da compreensão de código de forma aleatória.

Conclusão QP2: *os classificadores K-Nearest Neighbors, Neural Network, Naïve Bayes, Random Forest e Support Vector Machine apresentaram uma f-measure estatisticamente significante maior que um classificador Naive para classificar compreensão de código dos desenvolvedores baseados nos dados EEG. Em particular, os classificadores apresentaram, em média, uma diferença de média 134% maior em relação ao classificador aleatório. Esse resultado indica que os classificadores de machine learning não realizaram classificação da compreensão de código de modo aleatório.*

6.2.4 QP3: Qual é a efetividade dos classificadores de *machine learning* ao adicionar a CogEff no conjunto de dados?

A questão de pesquisa 3 investiga se a adição dos valores fornecidos pela CogEff no conjunto de treinamento implica em melhora na efetividade na classificação da compreensão de código pelas técnicas de *machine learning*. Para investigar esta questão, o mesmo processo e tratamento dos dados EEG foi seguido conforme a Seção 6.1.2. A exceção em relação esse processo, é que a partir dos dados EEG são calculados os valores através da abordagem CogEff, e estes dados também integram junto ao resultado das outras abordagens, o conjunto de dados para treinar as técnicas de *machine learning*. Especificamente, esses dados foram utilizados para treinar as técnicas *K-Nearest Neighbor* (KNN), *Neural Network* (NN) e *Random Forest* (RF). Estas técnicas de *machine learning* foram escolhidas porque alcançaram uma efetividade muita alta na análise da QP1 deste capítulo.

A Tabela 39 apresenta o resultado da efetividade das técnicas *K-Nearest Neighbor* (KNN), *Neural Network* (NN) e *Random Forest* (RF) para classificar compreensão de programas dos desenvolvedores através das métricas de macro precisão, macro *recall* e macro *f-measure*. Resultados acima de 90% foram realçados na Tabela 39.

F-measure. A Figura 27 apresenta os resultados da efetividade dos classificadores *K-Nearest Neighbor* (KNN), *Neural Networks* (NN) e *Random Forest* (RF) através da métrica *f-measure* descritos na Tabela 39. **Muito Alto.** A efetividade das técnicas treinadas com o conjunto de dados com as abordagens tradicionais EEG e CogEff permaneceu muito alta. Isto porque a *f-measure* dessas técnicas permaneceram com resultados acima de 80%. Especificamente, a efetividade dos classificadores *K-Nearest Neighbor* e *Neural Network* alcançou 90%, e a efetividade da técnica *Random Forest* alcançou uma *f-measure* de 87%.

A Figura 28 apresenta a distribuição da efetividade, baseado na métrica de *f-measure*, das técnicas de *machine learning* treinadas com as (A) Abordagens de EEG tradicionais e (B) Abordagens de EEG tradicionais e CogEff. A Figura 28 evidencia que os resultados da efetividade da *f-measure* das técnicas treinadas com (B) abordagens de EEG tradicionais e CogEff estão concentrados em valores mais altos em relação as técnicas treinadas com as (A) abordagens de EEG tradicionais.

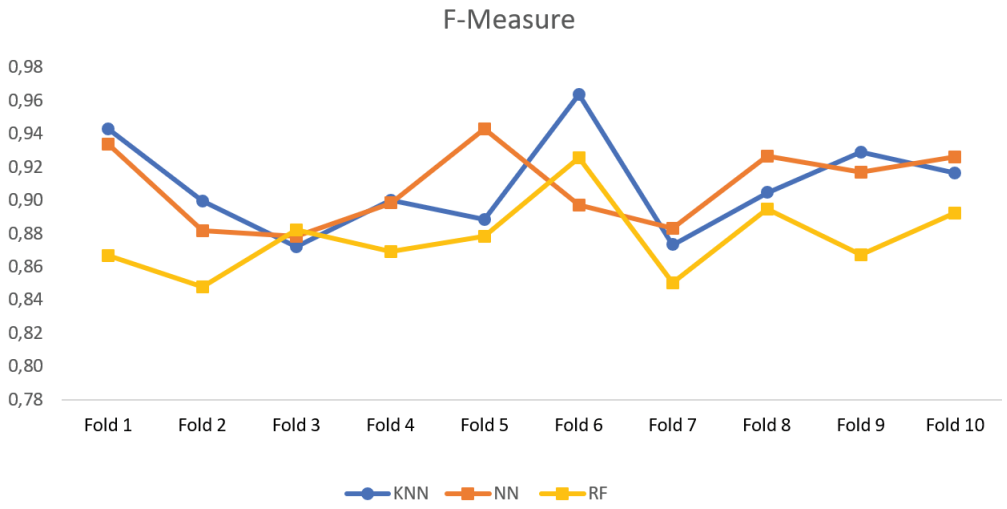
Tabela 39: Efetividade (*f-measure*) das técnicas de *machine learning* treinada com as abordagens EEG tradicionais e CogEff.

	KNN			NN			RF		
	Prec.	**Re.**	**F1-m**	**Prec.**	**Rec.**	**F1-m**	**Prec.**	**Rec.**	**F1-m.**
Fold 1	**0,95**	**0,94**	**0,94**	**0,93**	**0,93**	**0,93**	0,89	0,87	0,86
Fold 2	**0,90**	**0,90**	0,89	0,87	0,89	0,88	0,86	0,84	0,84
Fold 3	0,84	**0,90**	0,87	0,87	0,88	0,87	**0,90**	0,85	0,88
Fold 4	0,88	**0,92**	**0,90**	0,88	**0,91**	0,89	0,88	0,89	0,86
Fold 5	0,87	**0,90**	0,88	**0,95**	**0,93**	**0,94**	**0,90**	0,88	0,87
Fold 6	**0,97**	**0,95**	**0,96**	0,88	**0,91**	0,89	**0,91**	**0,93**	**0,92**
Fold 7	0,85	**0,90**	0,87	0,89	0,86	0,88	0,85	0,84	0,85
Fold 8	0,89	**0,91**	**0,90**	**0,92**	**0,93**	**0,92**	**0,90**	**0,92**	0,89
Fold 9	**0,93**	**0,93**	**0,92**	**0,91**	**0,92**	**0,91**	0,88	0,88	0,86
Fold 10	**0,93**	**0,91**	**0,91**	**0,93**	**0,92**	**0,92**	0,88	0,89	0,89
Todas	**0,90**	**0,91**	**0,90**	**0,90**	**0,91**	**0,90**	0,88	0,88	0,87

Legenda:
KNN: *K-Nearest Neighbor*, NN: *Neural Network*, RF: *Random Forest*,
Prec: Precisão, Rec.: *Recall*, F1-m: *F-Measure*

Fonte: Elaborado pelo autor.

Figura 27: Representação da efetividade das MLT treinadas com a CogEff.

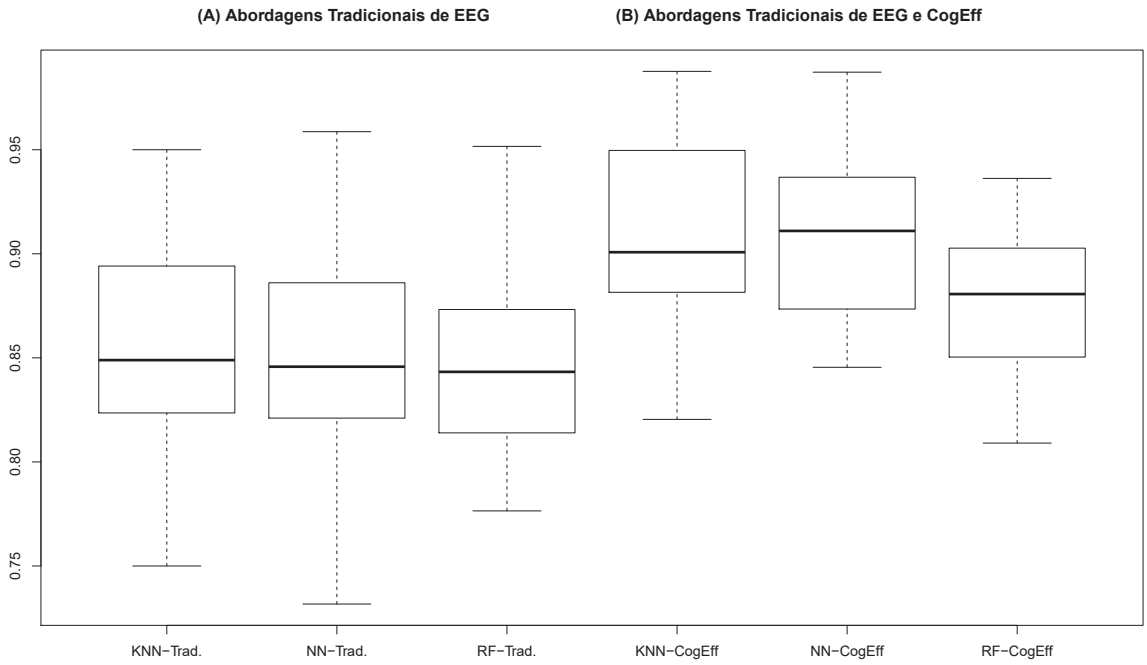


Fonte: Elaborado pelo autor.

Os valores mínimos dos classificadores treinados com o conjunto de dados A, possuem valores mínimos, e medianas menores em relação aos classificadores treinados com o conjunto de dados com a CogEff (B). Especificamente, os valores mínimos da *f-measure* dos classificadores KNN, NN e RF treinados com o conjunto de dados A são de 0,75, 0,73 e 0,77 respectivamente. Após os classificadores KNN, NN e RF serem treinados com o conjunto de dados B, os valores mínimos da *f-measure* são de 0,82, 0,84, e 0,80 respectivamente. Esses valores representam um aumento de 9,3% na técnica KNN, de 15% na técnica NN, e de 3,8% na técnica RF no valor mínimo da *f-measure*.

Esse aumento da efetividade também é evidenciado através dos valores da mediana da *f-measure* dos classificadores KNN, NN e RF. A mediana dos classificadores KNN, NN e RF

Figura 28: Representação da distribuição da efetividade das MLT treinadas com as (A) Abordagens tradicionais de EEG e (B) Abordagens tradicionais de EEG e CogEff.



Fonte: Elaborado pelo autor.

que foram treinados com o conjunto de dados (A) foi de 0,85, 0,85 e 0,84 respectivamente. A mediana dos classificadores KNN, NN e RF que foram treinados com o conjunto de dados (B) foi de 0,90, 0,91 e 0,88 respectivamente. Após treinar os classificadores KNN, NN e RF com as abordagens de EEG tradicionais e CogEff (B) as técnicas obtiveram um aumento da mediana de 5,9%, 7%, e 4,76% respectivamente.

A distribuição do resultado da efetividade de todos os classificadores foi testada com os testes de *Shapiro Wilk* e *Anderson Darling*. Esses testes evidenciaram que os dados pertencem a uma distribuição normal, onde $p\text{-}value > 0{,}05$. Por isso, foi realizado um *t-test* para verificar se a diferença entre a efetividade dos classificadores KNN, NN e RF treinados com as abordagens de EEG tradicionais (A) é estatisticamente significativa em relação aos mesmos classificadores treinados com as abordagens de EEG tradicionais e CogEff (B). Os testes estatísticos foram repetidos com o teste não paramétrico de Wilcoxon.

Testes estatísticos. Os resultados dos testes estatísticos são apresentados ambos na Tabela 40 e na Tabela 41. Esses testes verificaram se existe uma diferença significativa entre a efetividade das técnicas de *machine learning* KNN, NN e RF treinadas com as abordagens de EEG tradicionais (A) em relação as mesmas técnicas treinadas com abordagens de EEG tradicionais e CogEff (B). Especificamente, a Tabela 40 apresenta os resultados do *t-test*, enquanto a Tabela 41 apresenta os resultados do teste de Wilcoxon.

De acordo com os resultados do *t-test* apresentados na Tabela 40, existe uma diferença significativa entre a efetividade (*f-measure*) de todos os classificadores KNN, NN e RF treinados com as abordagens tradicionais de EEG (A) em relação aos classificadores que foram treina-

Tabela 40: Resultado do *t-test* entre a efetividade (*f-measure*) das MLTs treinadas com as abordagens tradicionais de EEG e das MLTs treinadas com as abordagens tradicionais de EEG e a CogEff.

	N	t	df	Média	Desvio Padrão	Intervalo de Confiança		*p-value*
						Inferior	Superior	
KNN Geral	30	-3,82	56,3	0,86	0,053	-0,074	-0,02	0,0003
KNN CogEff	30			0,9	0,044			
NN Geral	30	-4,65	51,4	0,85	0,054	-0,079	-0,031	<0,0001
NN CogEff	30			0,9	0,037			
RF Geral	30	-2,75	52,6	0,84	0,045	-0,048	-0,007	0,0079
RF CogEff	30			0,87	0,032			

Fonte: Elaborado pelo autor.

dos com as abordagens tradicionais de EEG e CogEff (B). Especificamente, foram obtidos os seguintes resultados utilizando o *t-test*.

KNN geral x KNN CogEff. A Tabela 40 apresenta que t(-3,82) = 56,3, $p\text{-}value = 0{,}0003$. A diferença entre as médias entre o classificador KNN treinado com as abordagens tradicionais de EEG (Geral) e o classificador KNN treinado com as abordagens tradicionais de EEG e a CogEff (KNN CogEff), e a direção do valor de t, pode-se concluir que existe um aumento significativo na efetividade para classificar a compreensão de código dos desenvolvedores ao utilizar o classificador KNN treinado com as abordagens tradicionais de EEG e CogEff. Especificamente, a média da *f-measure* apresentou aumento de $0{,}86 \pm 0{,}05$ para $0{,}90 \pm 0{,}04$. Um aumento de $0{,}04 \pm 0{,}02$ da *f-measure*.

NN geral x NN CogEff. A Tabela 40 apresenta que t(-4,65) = 51,4, $p\text{-}value < 0{,}0001$. A diferença entre as médias entre o classificador NN treinado com as abordagens tradicionais de EEG (Geral) e o classificador NN treinado com as abordagens tradicionais de EEG e a CogEff (KNN CogEff), e a direção do valor de t, pode-se concluir que existe um aumento significativo na efetividade para classificar a compreensão de código dos desenvolvedores ao utilizar o classificador NN treinado com as abordagens tradicionais de EEG e a CogEff. Especificamente, a média da *f-measure* apresentou um aumento de $0{,}85 \pm 0{,}05$ para $0{,}90 \pm 0{,}03$. Um aumento de $0{,}05 \pm 0{,}02$ da *f-measure*.

RF geral x RF CogEff. A Tabela 40 apresenta que existe uma diferença significativa entre os resultados da *f-measure* entre o classificador *Random Forest* treinados com as abordagens tradicionais de EEG e o classificador treinado com as abordagens tradicionais de EEG e CogEff, pois t(-2,75) = 52,6, $p < 0{,}0079$. A diferença entre as médias entre o classificador RF treinado com as abordagens tradicionais de EEG (RF Geral) e o classificador RF treinado com as abordagens tradicionais de EEG e a CogEff (RF CogEff), e a direção do valor de t, pode-se concluir que existe um aumento significativo na efetividade para classificar a compreensão de código dos desenvolvedores ao utilizar o classificador RF treinado com as abordagens tradicionais de EEG e a CogEff (RF CogEff). Especificamente, a média da *f-measure* apresentou um aumento de $0{,}84 \pm 0{,}04$ para $0{,}87 \pm 0{,}03$. Um aumento de $0{,}03 \pm 0{,}03$ da *f-measure*.

Teste de Wilcoxon. Os resultados do teste de Wilcoxon apresentados na Tabela 41 confir-

mam os resultados apresentados no *t-test* na Tabela 40.

Tabela 41: Resultado do teste de Wilcoxon entre a efetividade (*f-measure*) das MLTs treinadas com as abordagens tradicionais de EEG, e das MLTs treinadas com as abordagens tradicionais de EEG e a CogEff.

	N	Mediana	Média	Média de Ranks	Mediana das diferenças	Wilcoxon Rank	Intervalo de Confiança		*p-value*
							Inferior	Superior	
KNN Geral	30	0,84	0,86	0,86	-0,04	215	-0,07	-0,02	0,0005
KNN CogEff	30	0,9	0,9	0.91					
NN Geral	30	0,84	0,85	0.85	-0,05	179	-0,08	-0,03	<0,0001
NN CogEff	30	0,91	0,9	0,91					
RF Geral	30	0,84	0,84	0.85	-0,02	262	-0,05	-0,01	0,005
RF CogEff	30	0,88	0,87	0.88					

Fonte: Elaborado pelo autor.

A Tabela 41 apresenta que todos os classificadores treinados com abordagens EEG com a CogEff evidenciaram um aumento estatisticamente significativo em relação a um classificador treinado com dados psicofisiológicos. Especificamente, todos os testes apresentaram o *p-value* $<$ 0,05, e também evidenciaram valores de média e a mediana da *f-measure* maiores nos resultados dos classificadores treinados com um conjunto de dados psicofisiológicos com a CogEff.

Conclusão QP3: *os classificadores K-Nearest Neighbors, Neural Network e Random Forest obtiveram uma efetividade muita alta ($\geq 80\%$) para classificar compreensão de código baseado em um treinamento com um conjunto de dados que continham resultados de abordagens tradicionais de EEG e a CogEff. Esses resultados indicam maior efetividade desses classificadores para classificar compreensão de código em relação à aos mesmos classificadores que foram apenas treinados com as abordagens de EEG tradicionais.*

6.3 Discussão

Esta Seção apresenta uma discussão sobre o capítulo 5. A Seção 6.3.1 discute a influência dos dados de EEG na eficácia das técnicas de *machine learning*. A Seção 6.3.2 discute sobre as implicações da utilização de dados EEG em técnicas de *machine learning* para classificar compreensão de código. Finalmente, a Seção 6.3.3 apresenta os desafios adicionais do uso de EEG para classificar a compreensão do código.

6.3.1 Influência do tipo de dados nos resultados obtidos pelas técnicas de *machine learning*

Os dados de EEG são um tipo de dados de série temporal com alta dimensionalidade (COHEN, 2017; LOTTE et al., 2018). Essa alta dimensionalidade é uma consequência da coleta de dados a partir de 14 canais a uma taxa de amostragem de 256Hz. Desta forma, foram

coletados os dados EEG de quatorze regiões diferentes do escalpo, e cada canal produziu 256 linhas de dados de EEG por segundo de cada participante. Para fornecer tratamento igual aos classificadores foi aplicado uma Análise de Componentes Principais (PCA). O PCA serviu para reduzir a dimensionalidade dos dados. O PCA é útil para classificadores que não são apropriados para dados de alta dimensionalidade. Além disso, os resultados refletem um tratamento igual dos dados. Foram avaliadas as técnicas de aprendizado de máquina usando validação cruzada de 10-*folds*. Isto significa que os dados de treinamento e teste foram divididos em 10 partes, e cada parte compreendia em uma porção aleatória de 90% dos dados para treinamento e 10% para teste. As técnicas de aprendizado de máquina utilizadas nesta pesquisa possuem características diferentes para lidar com dados EEG e consequentemente isso pode afetar os resultados.

Os resultados na Tabela 36 mostram que a média da eficácia entre todos os *folds* atingiu a maior *f-measure* em técnicas que podem classificar em limites de decisão não lineares, ou seja, *K-Nearest Neighbors* (KNN) e *Neural Networks* (NN), com 86% e 85,3%, respectivamente. Este resultado é esperado porque os dados de EEG são conhecidos por não ter distribuição normal e não são estacionários (LOTTE et al., 2018). Dados de alta dimensionalidade, tais como o EEG pode comprometer os resultados nesses classificadores. A literatura reporta que o *K-Nearest Neighbor* (KNN) geralmente não é eficaz com dados de alta dimensionalidade (LOTTE et al., 2007).

A técnica *Neural Network* pode facilmente produzir classificações enviesadas pelo risco de *overfitting* na construção do modelo baseado em dados EEG. No entanto, a Análise de Componentes Principais (PCA) reduziu a dimensionalidade dos dados do EEG. Com isso, o risco de causar *overfitting* no treinamento da técnica *Neural Network* foi reduzida.

Além disso, o PCA reduziu o risco de afetar negativamente o desempenho da técnica KNN, a qual reconhecidamente não tem bom desempenho com dados de alta dimensionalidade (LOTTE et al., 2018, 2007). Além disso, a validação cruzada de 10 *folds* reduziu o risco de *overfitting* em qualquer técnica de aprendizado de máquina incluindo a técnica *Neural Network*. A *f-measure* da técnica *Neural Network* estava acima de 80% em todas os *folds*, evidenciando que os resultados não foram consequência de uma divisão de dados que propicie os bons resultados de classificação.

A técnica *Support Vector Machine* (SVM) classifica dados utilizando um método linear (GOODFELLOW et al., 2016). Neste trabalho, essa característica do SVM implicou em uma efetividade menor com uma média de *f-measure* menor aos obtidos pela técnica *K-Nearest Neighbors* e *Neural Networks*. Este resultado reafirma a observação anterior de que classificar a compreensão do código com base em dados de EEG é mais eficaz utilizando classificadores não lineares.

O *Support Vector Machine* (SVM) também é reconhecida pela literatura por não ser afetada pelo curso da dimensionalidade, ou seja, por projetar classificações representativas mesmo com pequenos conjuntos de dados de treinamento (LOTTE et al., 2018). Isso evidencia que a eficácia

da classificação da compressão do código pode ser maior mesmo utilizando um conjunto de dados pequeno. O SVM alcançou mais de 70% de *f-measure* na Tabela 36.

A técnica de aprendizado de máquina *Random Forest* (RF) realiza a classificação dos dados baseados em uma combinação de várias sub árvores de decisão (GOODFELLOW et al., 2016). A *Random Forest* atingiu 85% de *f-measure* na Tabela 36. A eficácia muito alta alcançada pela *Random Forest* com os dados de EEG indica que esta técnica de classificação encontrou um conjunto de árvores de decisão que relaciona o comportamento do EEG à compreensão do código do desenvolvedor. A literatura também reconhece que a técnica *Random Forrest* é eficaz para realizar classificações baseados em dados de EEG (LOTTE et al., 2018).

O classificador *Naïve Bayes* (NB) não alcançou alta efetividade para classificar a compreensão do código usando dados de EEG. Dois principais motivos podem causar essa baixa efetividade no *Naïve Bayes* para classificar compreensão de código utilizando dados EEG:

(1) *Naïve Bayes* exige que todas as variáveis de treinamento não tenham relação entre si, isto é, as variáveis independentes precisam ser independentes uma das outras (RISH, 2001). A literatura reconhece que os dados de cada canal EEG possuem relação. Dessa forma, a informação de um canal pode geralmente justificar o comportamento de outro canal.

(2) Os dados do EEG também não apresentam uma divisão clara entre os valores em classes distintas. Os dados de EEG coletados não possuem intervalos específicos que possam ser designados exclusivamente a uma classe. Dessa forma, os dados EEG não se enquadram em um dos teoremas apresentados em (RISH, 2001) para que o que *Naïve Bayes* produza resultados de alta efetividade.

6.3.2 Implicações

Os resultados neste capítulo apontam que o EEG combinado com técnicas de *machine learning* é uma maneira efetiva de classificar compreensão de código dos desenvolvedores. Esta abordagem ajudaria a encontrar quais trechos de códigos desenvolvedores não compreendem. Essa informação implicaria na construção de sistemas de recomendação.

Implicações da classificação da compreensão de código dos desenvolvedores na indústria de software: ambientes de desenvolvimento de software podem utilizar as informações sobre a compreensão do código dos desenvolvedores para atribuir tarefas compatíveis com o nível de compreensão individual. Atribuir tarefas que os desenvolvedores estão mais acostumados a compreender pode reduzir o tempo para resolução de *bugs* e, consequentemente, melhorar a produtividade (FRITZ; MÜLLER, 2016). A indústria de software também tem usado as métricas da *Sonarqube* para estimar o tempo para resolver problemas de qualidade de código. Em particular, o *Sonarqube* é uma plataforma que fornece informações sobre a qualidade do código (SONARQUBE, 2021). Portanto, a indústria também poderia integrar o status da compreensão do código dos desenvolvedores em plataformas como *Sonarqube* (SONARQUBE, 2021). Essa integração permitiria que a indústria usasse o status de compreensão do código dos de-

senvolvedores como um aspecto para estimar o tempo de conclusão das tarefas. Esse estado de compreensão também pode ser utilizado por desenvolvedores para indicar que o código deve ser simplificado e dividi-lo em partes menores. Outra aplicação da classificação da compreensão do código dos desenvolvedores, a indústria poderia utilizar o argumento de que a presença de *code smeells* (VIDAL; MARCOS; DÍAZ-PACE, 2016) pode estar relacionada a situações em que os desenvolvedores de software não compreendem o código-fonte. Por exemplo, a presença de um método longo que compreende a existência de um método com um número elevado de linhas pode provavelmente aumentar a carga cognitiva do desenvolvedor e comprometer a compreensão do desenvolvedor (GONÇALES; FARIAS, 2020). Outro exemplo, seria a existência de uma *God Class* no código-fonte, ou seja, uma classe que centraliza métodos e funcionalidades, pode prejudicar a compreensão dos desenvolvedores. Em ambos os casos, as técnicas que classificam compreensão de código podem ser utilizadas como um indicativo para simplificar o código-fonte.

Além disso, a utilização de métricas psicofisiológicas para fornecer suporte aos desenvolvedores e indicar aspectos de qualidade pode não se restringir ao EEG. A literatura da engenharia de software já evidenciou o potencial de informações oculares, frequência respiratória, e a atividade eletrodérmica podem integrar uma plataforma e ser utilizada em conjunto com métricas de software para melhorar a qualidade do software (MÜLLER; FRITZ, 2016) e a produtividade dos desenvolvedores (FRITZ; MÜLLER, 2016). Pesquisas em engenharia de software aplicaram conjuntos de dados consistindo em dados psicofisiológicos de desenvolvedores em técnicas de aprendizado de máquina e apresentaram potencial para classificar problemas de qualidade de código, nível de produtividade dos desenvolvedores e nível de dificuldade dos desenvolvedores. Essas informações, junto a informações estática do código-fonte podem melhorar a estimativas do projeto e reduzir o tempo de entrega.

Implicações no bem-estar dos desenvolvedores de software: Companhias de desenvolvimento de software utilizam métodos ágeis para entregar produtos em ciclos de forma incremental (BEHUTIYE et al., 2020). É prática dessas empresas em muitas vezes pressionar os desenvolvedores durante os estágios de desenvolvimento de software para concluir cada ciclo em um tempo reduzido (KHANAGHA et al., 2021). Isto pode não ser uma realidade caso desenvolvedores fiquem estagnados por muito tempo em uma mesma tarefa (MÜLLER; FRITZ, 2015). O classificador de compreensão do código do desenvolvedor baseado em EEG pode ser usado na indústria para indicar possíveis tarefas de desenvolvimento de software em que eles pudessem compreender e resolver. Desta forma, a indústria conseguiria identificar situações de compreensão de código para atribuir tarefas de compreensão mais fácil aos desenvolvedores. Isso implicaria que desenvolvedores poderiam ser mais produtivos, ao passo que reduzem o estresse. Além disso, classificar a compreensão do código dos desenvolvedores permitiria aos desenvolvedores ter conhecimento sobre qual tipo de código-fonte eles não entendem. Outro cenário é usar o estado de compreensão do código do desenvolvedor para indicar quais desenvolvedores devem fazer uma pausa durante o trabalho ou para solicitar suporte de outro membro

da equipe.

A dedicação em várias horas na revisão de um código fonte sem a correta compreensão pode causar frustração e, consequentemente, prejudicar o bem-estar e produtividade dos desenvolvedores (MÜLLER; FRITZ, 2015; ZÜGER; FRITZ, 2018). Pesquisadores em engenharia de software já treinaram técnicas de aprendizado de máquina com dados psicofisiológicos para classificar as emoções e progresso nas tarefas de codificação. Além de evidenciar o poder que dados psicofisiológicos, tais como EEG e métricas oculares, também mostra o potencial que contabilizar as perspectivas dos desenvolvedores podem melhorar o bem-estar dos desenvolvedores de software. A combinação do status de emoções e progresso a compreensão do código podem ser aplicados para indicar ou evidenciar que a frustração dos desenvolvedores por estar relacionada ao status de compreensão do código e o quanto isso afeta seu desempenho no ambiente de trabalho. Por exemplo, pesquisadores podem investigar se recomendar uma lista de tarefas com códigos-fonte que sejam mais fáceis de serem entendidos pelo desenvolvedor implicaria em menos frustração e, consequentemente, em melhor bem-estar dos desenvolvedores. Caso contrário, se a frustração e a falta de progresso não estiverem relacionadas à compreensão do código dos desenvolvedores, a indústria poderia usar essas informações como um alerta para que haja descanso dos desenvolvedores, ou de que a tarefa tem necessidade de mais colaboradores para ser concluída.

Implicações nos aspectos éticos sobre a utilização dos dados EEG pessoais dos desenvolvedores: os dados cerebrais, como qualquer outro dado psicofisiológico, são informações pessoais e confidenciais (MÜLLER, 2015). Portanto, os pesquisadores devem investigar sobre como garantir a privacidade dos desenvolvedores e participantes. Na literatura, essa preocupação implicou em tornar os dados demográficos dos participantes anônimos. Isto evitaria a revelação de aspectos psicofisiológicos de uma determinada pessoa (MÜLLER; FRITZ, 2015). Desta forma, o foco do uso dos dados deve ser de prestar suporte para cada participante sem revelar dados pessoais. A prática de tornar os dados dos participantes com rótulos anônimos é um padrão em pesquisas que utilizam dados psicofisiológicos na engenharia de software, e então os pesquisadores analisaram essas técnicas de produção de dados sem expor os dados pessoais dos desenvolvedores (ZÜGER; FRITZ, 2018; MÜLLER, 2015; MÜLLER; FRITZ, 2016). As empresas podem adotar a mesma prática com seus funcionários, focando no atendimento particular e específico, sem divulgação de seus dados pessoais. Seria uma implicação antiética vincular pontuações, fatores de desempenho e atribuir problemas no código diretamente a uma característica fisiológica dos funcionários. A indústria e a academia devem utilizar os dados de EEG para obter resultados positivos na qualidade do software combinando a experiência do desenvolvedor na resolução de tais atividades, e seu bem-estar. Portanto, seria uma implicação negativa usar este tipo de informação para desqualificar ou apontar aspectos negativos dos desenvolvedores.

Outro aspecto ético importante diz respeito à manipulação a longo prazo desse tipo de dado por empresas e institutos de pesquisa. Em particular, isso implica em uma discussão sobre

políticas e práticas em que a indústria deve adotar em relação ao armazenamento, acesso, e propósitos para o uso desse tipo de dados. Por exemplo, os desenvolvedores devem ter direito a acessar aos próprios dados psicofisiológicos gerados nas atividades de trabalho ou pesquisa, garantindo a propriedade e acessibilidade de seus próprios dados psicofisiológicos. As empresas e a indústria não devem ter permissão para fornecer ou vender dados pessoais a terceiros. Neste caso, os desenvolvedores devem garantir o direito de solicitar a qualquer momento a limitação da empresa, e de requisitar a remoção de seus registros dos dados psicofisiológicos. Empresas terceirizadas podem usar os dados psicofisiológicos para verificar o estado de mental de desenvolvedores de software para outras finalidades comerciais. Neste caso, quaisquer dados fornecidos a terceiros devem exigir a consentimento de desenvolvedores. Na academia, os participantes geralmente assinam um termo de consentimento autorizando a coleta e armazenamento de dados para fins de pesquisa.

6.3.3 Desafios futuros

Limitações de sensores de EEG Comerciais. As informações dos sensores EEG têm potencial para integrar o ambiente de desenvolvimento de software e prestar suporte aos desenvolvedores (CRK; KLUTHE, 2016; KOSTI et al., 2018; LEE et al., 2017; FRITZ; MÜLLER, 2016; MÜLLER; FRITZ, 2016). Pesquisas em engenharia de software têm evidenciado a utilidade dos sensores do EEG para classificar a dificuldade das tarefas (KOSTI et al., 2018), o nível de conhecimento (LEE et al., 2017), as emoções e o progresso dos desenvolvedores (MÜLLER; FRITZ, 2015). Para coletar dados de EEG de desenvolvedores ou participantes em estudos anteriores, pesquisadores utilizaram EEG portáteis contendo até 14 canais de EEG (CRK; KLUTHE, 2016; CRK; KLUTHE; STEFIK, 2015). A principal vantagem desses dispositivos é que não são invasivos como a tecnologia de iEEG, que consiste na implantação intracraniana dos canais EEG (LACHAUX et al., 2012). O EEG convencional, em relação ao iEEG possui uma resolução de tempo menor. No entanto, o EEG permite rápida implantação nos desenvolvedores e é aplicável na indústria de software. O EEG portátil no qual os pesquisadores de engenharia de software usaram é uma versão simplificada das versões utilizadas em clínicas (Emotiv Systems, 2021). A indústria de hardware projetou esses equipamentos para usuários finais, e pesquisas de pequena escala. O primeiro problema é que no cenário industrial, esses dispositivos ainda são difíceis de implantar nos desenvolvedores por causa da quantidade de eletrodos e a facilidade com que os sensores perdem contato com o escalpo. Outro problema, é que não é simples posicionar os canais na região correta do escalpo o que obriga os desenvolvedores ter um auxílio para colocar o dispositivo. Outra preocupação é que boa parte das pesquisas em engenharia de software utiliza dispositivos com menos de 14 canais (ZÜGER; FRITZ, 2018; FRITZ; MÜLLER, 2016). A consequência é uma resolução espacial menor, consequência da menor cobertura da atividade cerebral captada pelos canais EEG.

Para superar esses desafios, a academia e a indústria poderiam propor dispositivos de EEG

adequados para ambientes de desenvolvimento de software. Pesquisadores e a indústria de software poderiam propor soluções mais fáceis de implantar nos participantes. Os dispositivos devem conter mecanismos que permitam aos uma fácil usabilidade por parte de desenvolvedores de softwar. Isso economizaria tempo para iniciar as atividades de experimentos e não seria um fator que desencorajaria os desenvolvedores de utilizar esses dispositivos (RADEVSKI; HATA; MATSUMOTO, 2015). A indústria poderia melhorar a composição de matérias primas utilizadas nos canais de EEG. Atualmente, os sensores são compostos por esponjas que secam rapidamente e podem perder o contato rapidamente, impossibilitando novas leituras das ondas cerebrais. A solução poderia ser trocar a esponja por um material que não drene tão rápido. Outra melhoria é desenvolver um dispositivo com um número maior de sensores, por exemplo, entre 128 e 256 canais, para uso na indústria. Isso viabilizaria a produção de dados com maior cobertura da atividade cerebral.

Filtragem e processamento de dados psicofisiológicos. O processamento e a filtragem dos sinais de EEG são etapas importantes para garantir a qualidade dos dados (CHEVEIGNÉ; NELKEN, 2019). Os pesquisadores e a indústria de software devem aplicar os procedimentos adequados de processamento e filtragem nos sinais de EEG para isolar o sinal de interesse e realizar a análise desejada. Por causa da falta de estudos que propusessem filtros e etapas para processamento e filtragem de dados de EEG para aplicações na engenharia de software, pesquisadores utilizaram etapas de processamento e filtragem adaptados de estudos de aplicação geral do EEG (KOSTI et al., 2018; DURAISINGAM; PALANIAPPAN; ANDREWS, 2017; MÜLLER; FRITZ, 2016; CRK; KLUTHE, 2016), processo que está especificado na Seção 6.1.2. No entanto, pesquisadores e indústria tiveram que adaptar filtros e procedimentos de acordo com sua preferência sem avaliar o impacto da implementação. Existem várias técnicas para filtrar sinais de EEG, tais como a utilização de filtros FIR e IIR (CHEVEIGNÉ; NELKEN, 2019; PARKS; BURRUS, 1987). Além disso, os pesquisadores podem utilizar diferentes tipos de janelas para processar a filtragem, tais como as janelas de *Hanning* e *Hann*. Os filtros também podem atuar nas bandas *low-pass*, *high-pass*, e *stop-bands*. Pesquisadores têm adotado o *design* FIR em filtros *high-pass* e *low-pass* de sinais de EEG na engenharia de software. Além disso, outra etapa indefinida é o processamento de sinais de EEG. Alguns pesquisadores decompõem o sinal EEG em cinco bandas e normalizam esses valores. Além disso, pesquisadores também escolhem aleatoriamente quais bandas, tais como alfa (α) e theta (θ), ou até mesmo um conjunto de canais específicos de EEG (CRK; KLUTHE, 2016).

Para definir um filtro padrão, os pesquisadores poderiam investigar qual tipo de filtro implica no sinal de interesse, ou seja, que remove ruídos e componentes externos que não são de interesse. Na prática, os pesquisadores devem realizar estudos experimentais para investigar qual filtro é adequado para sinais de EEG em estudos de engenharia de software (GONÇALES et al., 2021). Existem muitos fatores que podem afetar os sinais de EEG com ruídos no sinal, tais como o ambiente, e diferentes sinais vitais incorporados no EEG. Os pesquisadores podem comparar a eficácia dos filtros em sinais de EEG com sinais não filtrados em relação aos sinais

filtrados. Por exemplo, os pesquisadores podem treinar modelos de aprendizado de máquina para classificar a dificuldade das tarefas dos desenvolvedores medindo métricas de efetividade, como precisão e *recall*. O que se espera é que o sinal filtrado com o filtro ideal seja mais efetivo em termos de precisão e *recall*. Outro exemplo é um estudo experimental onde os sinais filtrados de EEG podem apresentar uma correlação mais clara com a compreensão de código. Experimentos controlados podem ser usados para testar o conjunto adequado de variáveis independentes que podem ser extraídas dos sinais de EEG. Para isso, pesquisadores poderiam testar uma série de ferramentas de correlações estatísticas, como a correlação de *Pearson* e *Spearman*, e ferramentas para seleção de variáveis independentes contidos em bibliotecas de software, tais como *Weka* (EIBE; HALL; WITTEN, 2016; HALL et al., 2009) e *ScikitLearn* (PEDREGOSA et al., 2011).

Conhecimento empírico sobre funções cerebrais durante compreensão de código utilizando sinais de EEG. Pesquisadores em engenharia de software tem investigado a ativação de regiões do cérebro envolvidas na compreensão de código. Essas pesquisas têm evidenciado que várias áreas do cérebro são ativadas durante a compreensão do código (SIEGMUND et al., 2017), como as responsáveis pelo processamento da linguagem, atenção, e memória de trabalho (PEITEK et al., 2018a). SIEGMUND et al. (2014), e CASTELHANO et al. (2019) evidenciaram que áreas do cérebro relacionadas à matemática também podem ser ativadas durante a compreensão do programa. Além das contribuições de SIEGMUND et al. (2014) e CASTELHANO et al. (2019) na investigação de compreensão de programas em relação às atividades cerebrais, os pesquisadores poderiam continuar contribuindo nesta área com a finalidade de confirmar os resultados de CASTELHANO et al. (2019) e SIEGMUND et al. (2014). Pesquisas também devem ser conduzidas para estender e encontrar possíveis novas áreas ou possíveis resultados que podem discordar das pesquisas anteriores. Outra preocupação é que os pesquisadores realizaram análises e correlacionaram as áreas do cérebro a tarefas de compreensão de código usando o fMRI. O fMRI fornece alta precisão espacial e apresenta em imagem a ativação que ocorre nas áreas profundas do cérebro. Desse modo, também seria desejável que pesquisas fornecessem a relação das áreas cerebrais relacionadas à compreensão do código por meio do EEG. Esse tipo de pesquisa forneceria base teóricas para as pesquisas sobre compreensão de código que usam dispositivos de EEG.

Para superar esses desafios, os pesquisadores poderiam se concentrar na produção de novos conhecimentos empíricos sobre a relação das funções cerebrais e tarefas de compreensão de código. Os pesquisadores podem usar experimentos controlados para desenvolver esses estudos. Os pesquisadores poderiam usar o fMRI para estender a literatura existente para reforçar os resultados atuais para um conjunto mais amplo de tarefas e com um número maior de participantes (PEITEK et al., 2018a; SIEGMUND et al., 2017; SIEGMUND, 2016; SIEGMUND et al., 2014). Para resolver o problema da ausência do uso de sensores de EEG para correlacionar a compreensão do código e a localização de ativação de áreas cerebrais, os pesquisadores poderiam replicar os estudos existentes que utilizam o fMRI, mas com os dispositivos de de

EEG. Para isso, técnicas de localização de fonte de imagem podem ser utilizadas para identificar as áreas ativas no cérebro enquanto os desenvolvedores realizam tarefas de compreensão (MICHEL; HE, 2019).

Utilização dos resultados deste trabalho considerando questões éticas e privacidade na prática. Embora este trabalho tenha apresentado a importância de considerar questões éticas e de privacidade dos usuários, aplicar isso na prática se torna desafiador desde que as empresas, a princípio, poderiam argumentar o gasto investido para coletar dados de EEG de desenvolvedores para comercializar esses dados. Outra desconfiança que emerge da utilização de dados biométricos no ambiente de trabalho, é que seria difícil acreditar que empresas não comercializem os dados pessoais dos desenvolvedores, já que se tratam, de um tipo de dado que a qualquer maneira, possui alto valor agregado para várias áreas de aplicação. A partir da obtenção indevida aos dados, outras empresas poderiam procurar desenvolver outras aplicações e tecnologias para o setor, ou até mesmo, recomendar serviços as vítimas baseadas em seus próprios dados biométricos de forma não autorizada, tais como, remédios, ou serviços psicológicos. Portanto, gerenciar os dados biométricos dos desenvolvedores de modo ético, e mantendo a privacidade, é um desafio relevante para essa área.

Com base nisso, três possíveis maneiras de como as empresas no futuro podem aplicar na prática os resultados deste trabalho respeitando questões éticas e de privacidade: (1) As leituras dos dados psicofisiológicos dos usuários poderiam ficar em posse dos desenvolvedores durante todo o tempo. Desta forma, dados seriam armazenados em discos ou dispositivos de armazenamento de propriedade do próprio desenvolvedor. Aplicações do ambiente de trabalho teriam apenas permissão de leitura para realizar as recomendações aos desenvolvedores. (2) Uso de *blockchain* junto aos dados psicofisiológicos. O *blockchain* é utilizado no sistema financeiro de criptomoedas e é um mecanismo muito efetivo para rastrear sucessivas operações financeiras de certo ativo (DEEPA et al., 2022). Consequente, identifica a origem da operação. Essa característica pode ser relevante para os dados psicofisiológicos dos usuários, pois ajudaria a confirmar a origem dos dados. (3) Engenharia social para assegurar os direitos a privacidade e garantir diretrizes éticas ao lidar com dados psicofisiológicos dos desenvolvedores de software. As vezes nem sempre uma solução tecnológica é efetiva para garantir privacidade e direitos dos usuários nas aplicações em que eles fazem parte. Para isso, garantir proteção na esfera legal para os desenvolvedores também seria necessário. Reparo a imagem do desenvolvedor com implicações de perda financeira a empresa poderia ser de relevante ajuda a não incentivar um uso indevido dos dados.

6.4 Ameaças a Validade

Esta seção descreve as medidas que foram realizadas para mitigar aspectos que ameaçam a validade deste trabalho. Portanto, foram mitigadas as ameaças aos aspectos externos, internos e validade de construção (WOHLIN et al., 2012), as quais estão descritas abaixo.

Validade Interna. Essa validade se refere a ações realizadas para mitigar riscos que ameacem a validade dos dados usados neste estudo. Os sinais de EEG contêm vários componentes não relacionados ao objetivo deste capítulo, tais como os movimentos musculares e oculares. Além disso, os sinais podem conter ruídos externos, tais como interferência de sinal de vários aparelhos eletrônicos. Para isso, foram aplicados filtros EEG para atenuar ruídos e componentes indesejados da análise. Apesar disso, ainda é possível que contenha ruídos e interferências nos sinais de EEG. Nesta tese foi utilizada a biblioteca *Python MNE* para realizar a análise de componentes independentes. Além disso, para preservar o objetivo do estudo, foram utilizados apenas dados do EEG para treinar os classificadores de *machine learning*.

Validade a construção. Essa validade se refere às medidas tomadas para mitigar riscos que podem afetar a confiabilidade dos dados. Os dados possuem marcadores específicos para indicar a localização das tarefas nas leituras dos sinais de EEG. Este capítulo adotou diferentes marcadores que identificam se o desenvolvedor compreendeu o código de modo correto, incorreto ou não respondeu. Foram realizadas duas medidas para evitar que os participantes ganhassem experiência em tarefas que pertencem ao mesmo grupo. Para isso, os dados do *dataset* foram coletados da seguinte forma: (1) randomizou a sequência em que as tarefas de compreensão para os participantes; (2) a estrutura e a lógica entre as tarefas foram diferentes, mesmo que tenham o mesmo propósito ou pertença ao mesmo grupo.

Ameaça externa. Refere-se às medidas adotadas para mitigar riscos de generalização dos resultados. Os pesquisadores podem generalizar os resultados deste estudo para o ambiente de laboratório porque as tarefas consistem em tarefas de compreensão, onde participantes devem realizar uma leitura do código fonte, e deduzir um resultado do algoritmo apresentado. Além disso, alguns aspectos podem limitar a reprodução desses resultados em laboratório. Os dados do *dataset* contém resultados dos classificadores de compreensão de código limitados a 10 tarefas em Java, com 60 segundos de duração. O fator humano também pode influenciar na generalização dos resultados, devido às personalidades diferentes. Portanto, foram recrutados de participantes com perfis semelhantes para manter a homogeneidade dos dados da amostra. No geral, os participantes têm experiencia em desenvolvimento de software em um intervalo que varia de 2 a 7 anos. Assim, a diferença entre os níveis de experiência não foi significativa para impactar nos sinais de EEG. Por fim, efeitos de aspectos externos tais como níveis de stresse, e interrupções não foram considerados na avaliação. Nenhum dado em relação a stress e interrupções não estava presente no conjunto de dados utilizado. Portanto, stress pode ser um fator que possa impor risco aos resultados. Porém, esses fatores foram atenuados em relação as características do ambiente utilizado. Participantes conduziram a tarefa de modo individual, e além disso, o experimento fornecia períodos de relaxamento entre as tarefas. Essas medidas atenuam as interferências externas em relação ao stress e as interrupções. Por fim, acredita-se ser desafiador reproduzir os mesmos resultados obtidos nesta seção, e para que uma repetição dos resultados ocorra com a maior precisão possível, é necessário coletar os dados ao menos com essas mesmas medidas em relação ao ambiente, e recrutar participantes com perfis semelhantes

ao contidos no conjunto de dados.

6.5 Considerações finais

Este capítulo avaliou a efetividade do uso de técnicas de aprendizado de máquina para classificar a compreensão de código baseados em dados de EEG. Esta análise foi necessária principalmente pela falta de classificadores de compreensão de código baseados em EEG pelas pesquisas em engenharia de software, os quais focaram em outros propósitos tais como, o nível de dificuldade e de conhecimento dos desenvolvedores.

Os resultados mostraram que é possível usar técnicas de *machine learning* para classificar a compreensão de código dos desenvolvedores com base em dados de EEG. As técnicas *K-Nearest Neighbors*, *Neural Network* e *Random Forrest* apresentaram uma *f-measure* significativamente maior que 80% para classificar a compreensão do código dos desenvolvedores com base nos dados de EEG. Além disso, as técnicas *K-Nearest Neighbors*, *Neural Network*, *Random Forrest*, *Support Vector Machine* e *Naïve Bayes* possuem efetividade significativamente maior que um classificador randômico. Por fim, a adição dos resultados da CogEff no conjunto de dados aumento significativamente a efetividade dos classificadores *K-Nearest Neighbors*, *Neural Network* e *Random Forrest*.

Um possível trabalho futuro é analisar os efeitos de diferentes janelas de tempo na precisão, *recall* e *f-measure* das técnicas de *machine learning*. Pretende-se realizar um estudo experimental através da aplicação do questionário TAM (*Technology Acceptance Model*) para identificar aceitabilidade, usabilidade, utilidade percebida, intenção de uso em relação ao uso de dados de EEG em tempo real na área de engenharia de software. Além disso, outros fatores podem afetar a precisão dos classificadores e pode ser investigado futuramente: classificação de compreensão de código utilizando tarefas de compreensão que são desconhecidas pelo classificador, e com a utilização de dados EEG de um novo participante, ou uma combinação de ambos. A inserção de dados online obtidos dos participantes nas técnicas de *machine learning* também pode impactar na precisão dos classificadores. Por fim, estudos futuros poderiam considerar fatores externos junto a dados de EEG para treinar classificadores de *machine learning* para classificar a compreensão de código de desenvolvedores. O próximo capítulo apresenta a conclusão desta tese com uma sumarização dos principais resultados, as contribuições concretizadas, e dos trabalhos futuros.

7 CONCLUSÃO

Esta tese investigou problemas e limitações da literatura contemporânea sobre o tema de carga cognitiva na área de engenharia de software, particularmente em relação à correlação das abordagens baseadas em EEG com a compreensão de software. Para isso, uma questão de pesquisa geral (*QPGeral*) foi elaborada para a investigação deste problema: "*Qual é a relação entre abordagens baseadas em EEG utilizadas para mensurar a carga cognitiva com tarefas de compreensão de software?*" Por se tratar de uma questão ampla, outras quatro questões de pesquisa foram definidas visando resolver os problemas específicos (Capítulo 1.1). Portanto, o objetivo principal desta tese foi investigar a viabilidade de medir carga cognitiva nas tarefas de compreensão de código através de abordagens de EEG. Uma série de resultados foram obtidos. Baseados nesses resultados, uma sumarização foi realizada deste trabalho junto às respectivas considerações finais e futuros desafios.

Este Capítulo é organizado em três partes: a Seção 7.1 descreve a sumarização e principais conclusões desta pesquisa; a Seção 7.2 apresenta as contribuições obtidas após os resultados. Por fim, a Seção 7.3 descreve os futuros desafios, os quais foram extraídos ao longo da investigação desta tese.

7.1 Sumário

Mensurar a carga cognitiva dos desenvolvedores é importante porque possibilita considerar o aspecto humano nas tarefas de engenharia de software. Ao contrário de abordagens que se baseiam em aspectos estáticos do código fonte, tais como, a quantidade de linhas, ou quantidade métodos, a carga cognitiva é uma característica individual dos desenvolvedores que pode ser coletada e mensurada durante a condução das tarefas relacionadas a engenharia de software, tais como a compreensão de código. Desta forma, a carga cognitiva tem um potencial preditivo maior em relação às abordagens tradicionais, pois, estas apenas podem ser consolidadas após a conclusão das tarefas de compreensão de código. Por causa disso, a carga cognitiva foi explorada em pesquisas na área de engenharia de software. Apesar de ser extensamente aplicada e empregada nas pesquisas de engenharia de software, pesquisadores mensuraram a carga cognitiva sem se basear ou analisar a correlação das abordagens utilizadas no contexto que foram analisados. O principal problema que isso implica é a adoção de abordagens com o risco de não serem apropriadas para os propósitos investigados, e de não explorarem o potencial das abordagens corretas para o propósito explorado. Para resolver esses problemas, uma investigação na literatura em relação à carga cognitiva na engenharia de software foi necessária e uma nova solução para mensurar carga cognitiva em tarefas de compreensão de código baseado em dados EEG foi desenvolvida. Além disso, testes de correlação, e treinamento de técnicas de *machine learning* com dados da carga cognitiva de desenvolvedores de software para classificar compreensão de código.

A primeira parte do estudo consistiu em classificar e obter uma visão geral sobre as medidas da carga cognitiva na área de engenharia de software (QP1). Essa investigação na literatura foi necessária pois faltavam estudos de mapeamento sistemáticos que sumarizassem o estado da arte sobre a carga cognitiva na engenharia de software para apontar possíveis desafios existentes. Mas, as revisões da literatura que foram realizadas se limitavam geralmente a algum tipo de dispositivo para coletar dados relacionados a carga cognitiva, tal como o *Eye-Tracker* ou focado em alguma área específica, ou tratavam sobre iteração-cérebro computador em áreas gerais, sem tratar as especificidades da carga cognitiva na área de engenharia de software. Além disso, nenhum desses estudos utilizou um protocolo de revisão de mapeamento sistemático. Por isso um estudo de mapeamento sistemático foi conduzido com objetivo de fornecer uma visão geral em relação às medidas de carga cognitiva na engenharia de software. Esse protocolo definiu uma *string* de busca baseado na literatura de interesse, os motores de buscas para pesquisar por estudos em repositórios com estudos relacionados a engenharia de software e carga cognitiva, definição de questões de pesquisa em relação aos aspectos da pesquisa, definição de critérios de extração e exclusão relacionados baseados nas questões de pesquisas, um processo com 8 passos para filtrar os estudos primários, e o detalhamento de um esquema para extração dos dados dos estudos selecionados. Essa filtragem possibilitou a seleção de 63 estudos primários a partir de 4,175 artigos iniciais obtidos a partir da inserção da *string* de busca nos respectivos motores de busca.

Desta forma, respondendo a questão de pesquisa 1 desta tese, o estado da arte em relação às medidas da carga cognitiva na área de engenharia de engenharia de software é a seguinte: (1) cerca de 41% dos estudos primários (20/63) não definiram o conceito de carga cognitiva; (2) 43% dos estudos primários (27/63) utilizaram dispositivos multimodais para medir a carga cognitiva, isto é, combinaram diferentes dispositivos tais como o EEG, e o *Eye-Tracker*; (3) 62% dos estudos primários (39/63) utilizaram métricas multimodais, isto, é extraíram mais de uma métrica dos dispositivos utilizados para capturar os dados dos desenvolvedores; (4) 11% dos estudos primários (7/63) adotaram múltiplas técnicas de *machine learning*, tais como, *Support Vector Machine* (SVM), *Neural Networks* (NN), *K-Nearest Neighbor* (KNN) para avaliar a efetividade aplicada a vários propósitos na engenharia de software; (5) 38% dos estudos primários (24/63) investigaram a carga cognitiva com o propósito de analisar a compreensão de código; (6) 83% dos estudos primários (52/63) utilizaram tarefas de programação para avaliar a carga cognitiva dos desenvolvedores; (7) 81% dos estudos primários (51/63) utilizaram código fonte como artefato para as avaliações da carga cognitiva; (8) 32% dos estudos primários (20/63) recrutaram entre 11 a 20 participantes para executar o estudo sobre a carga cognitiva; (9) 83% dos estudos primários (51/63) utilizam a metodologia de pesquisa de estudo de validação, este tipo de método refere-se a estudos que foram avaliados na academia tais como experimentos controlados, e que não foram avaliados na indústria; (10) 65% dos estudos primários (41/63) foram publicados e divulgados em conferências.

Após classificar os estudos primários e obter uma visão geral das medidas da carga cogni-

tiva na engenharia de software, o próximo passo foi desenvolver uma abordagem baseada em dados EEG para mensurar a carga cognitiva específica para compreensão de código(QP2). A abordagem de EEG para mensurar a carga cognitiva em tarefas de compreensão de código foi desenvolvida por causa de dois principais motivos: primeiro, as pesquisas em engenharia de software utilizaram uma gama de dispositivos para mensurar a carga cognitiva, e nesta pesquisa, foi escolhido o dispositivo que coleta dados de ondas cerebrais que possuem relação com a dinâmica neural. Dentre as opções para este tipo de dado, o EEG é o aparelho que tem potencial de ser realmente adotado na indústria pois é portável, e permite mobilidade do usuário, e, possui um tempo de resposta instantâneo aos estímulos, pois possui alta-precisão temporal, ao contrário de dispositivos do tipo fMRI ou fNRIS. O segundo motivo, é por causa da adaptação das abordagens de EEG para mensurar a carga cognitiva em tarefas de compreensão de código. Faltaram estudos tentando adaptar e criar uma abordagem específica, que fosse apropriada para essa finalidade. Alguns estudos reconhecendo essa necessidade se limitaram de modo arbitrário a restringir alguns aspectos, tais como, as frequências de bandas, e canais de EEG, e tipos de filtragem de sinais EEG específicos, as quais foram adotadas com base em julgamentos isolados dos pesquisadores. Além disso, esses aspectos foram divergentes entre os estudos. Desta forma, existe o risco de que as abordagens utilizadas para mensurar a carga cognitiva na compreensão de código não estivesse adequada. Por isso, para responder à questão de pesquisa 2 (QP2) foi desenvolvida a CogEff, uma abordagem para mensurar a carga cognitiva em tarefas de compreensão de código. Essa abordagem define um processo de três etapas, na qual a primeira etapa consiste na definição de um filtro com características específicas para mensurar carga cognitiva em tarefas de compreensão de código, a segunda etapa consiste na montagem de um grafo que representa a relação entre os canais EEG. As conexões entre os canais EEG foram estabelecidas utilizando uma abordagem *over-time*. Propriedades foram definidas para estabelecer a relevância dessas relações com a o contexto da compreensão de código. Por fim, a densidade do grafo é o resumo do valor final da CogEff.

A próxima questão de pesquisa da tese concentrou-se em avaliar a abordagem CogEff desenvolvida neste trabalho (QP3). Esta avaliação serve para analisar se a abordagem proposta tem relação com a compreensão de código, a qual é a tarefa em que a CogEff tem o objetivo de medir a carga cognitiva. Além disso, mesmo entre as pesquisas que utilizaram o dispositivo de EEG, trabalhos divergiram na abordagem utilizada para mensurar a carga cognitiva em tarefas da engenharia de software. Por exemplo, esses trabalhos utilizaram vários tipos de abordagens tais como a adoção das frequências de bandas alfa (α), beta (β), delta (δ), e theta (θ), ERD e ASR. Essas abordagens foram adotadas sem qualquer tipo de avaliação de correlação em relação ao propósito sendo investigado, entre eles, a compreensão de código. Por isso, a necessidade de avaliar a correlação dessas abordagens com compreensão de código. Em particular foi avaliado a correlação entre as abordagens EEG com dados categóricos das respostas das tarefas de compreensão, e com o tempo decorrido para concluir a tarefa de compreensão. Além disso, foi testado se a CogEff demonstra diferença entre tarefas de compreensão com

complexidades diferentes. Desta forma, respondendo a questão de pesquisa 3 (QP3), foi obtido que (1) os testes de *Kendall* e *Spearman* mostraram que a CogEff possui correlação negativa, de -2893 e -3537 respectivamente, com as respostas das tarefas de compreensão de código, enquanto que as ondas alfa (α) e beta (β) apresentaram correlações positivas com as respostas de compreensão de código; (2) os testes de *Pearson* e *Spearman* apresentaram que a CogEff possui correlação positiva, de 0,3742 e 0,1166 respectivamente, com o tempo decorrido para concluir as tarefas de compreensão. As abordagens a alfa (α), beta (β), delta (δ) e theta (θ) apresentaram correlação negativa em relação ao tempo gasto para completar as tarefas de compreensão de código; (3) houve diferença significativa nos valores da CogEff entre tarefas de complexidade maior e menor.

A última questão de pesquisa (QP4) investigou a efetividade das abordagens de EEG para treinar técnicas de *machine learning* com abordagens EEG para classificar compreensão de código. Isto era necessário pois as pesquisas na área de engenharia de software não avaliaram a efetividade para classificar a compreensão de código, mas para propósitos relacionados, tais como nível de dificuldade, e nível de conhecimento. Além disso, nenhuma análise verificou o quão distante a efetividade das técnicas de *machine learning* treinados com abordagens EEG são em relação à classificadores randômicos, o que coloca em dúvida, a real efetividade desses classificadores. Além disso, foi investigado a efetividade de acrescentar a abordagem desenvolvida neste trabalho, específica para mensurar carga cognitiva em tarefas de compreensão de código, nas técnicas de *machine learning* para classificar compreensão de código. Respondendo a última questão de pesquisa desta tese (QP4), a efetividade de utilizar abordagens EEG para classificar compreensão de código é a seguinte: (1) *K-Nearest Neighbors*, *Neural Network*, e *Random Forest* apresentaram um *f-measure* acima de 80% para classificar compreensão de código dos desenvolvedores baseados em dados EEG; (2) Todos os classificadores treinados com abordagens EEG apresentaram uma *f-measure* estatisticamente significante maior que o classificador aleatório para classificar compreensão de código dos desenvolvedores baseados nos dados EEG; (3) A efetividade dos classificadores *K-Nearest Neighbors*, *Neural Network* e *Random Forest* treinados com abordagens EEG tradicionais e CogEff apresentaram uma diferença significativa em relação a técnicas de *machine learning* treinados apenas com as abordagens de EEG tradicionais.

7.2 Contribuições

Esta seção descreve as contribuições que foram concretizadas ao longo do desenvolvimento desta tese. A seguir as contribuições são destacadas em relação as questões de pesquisa investigadas.

(1) Visão panorâmica sobre as medidas de carga cognitiva na área de engenharia de software: foi realizado um mapeamento sistemático que classificou o estado arte sobre as medidas da carga cognitiva na engenharia de software. O mapeamento sistemático contribuiu com

(1) uma classificação dos estudos sobre medidas da carga cognitiva na área de engenharia de software de acordo com 10 aspectos, em particular, sensores, métricas, técnicas de *machine learning* treinadas com dados psicofisiológicas, propósitos da utilização da carga cognitiva na engenharia de software, tipos de artefatos, tarefas que foram utilizados para mensurar a carga cognitiva, participantes, os métodos de pesquisa, e os locais de publicação; (2) um esquema de classificação para categorizar pesquisas sobre as medidas da carga cognitiva na engenharia de software, o qual foi derivado a partir das categorias das repostas; (3) uma relação de 63 estudos sobre as medidas da carga cognitiva na área de engenharia de software através de dispositivos psicofisiológicos; (4) uma descrição contribuições realizadas pelos estudos sobre mensurar a carga cognitiva na engenharia de software; (5) um levantamento dos desafios mais relevantes identificados nos estudos sobre mensurar a carga cognitiva na área de engenharia de software.

(2) Abordagem CogEff baseados em dados EEG para mensurar carga cognitiva de desenvolvedores de software em tarefas de compreensão de código: esta pesquisa contribuiu com o desenvolvimento de uma abordagem CogEff que mensura a carga cognitiva de desenvolvedores durante tarefas de compreensão de código. A abordagem CogEff consiste em três etapas, as quais são responsáveis por (1) filtrar sinais EEG, (2) estipular a interação e relação de canais EEG através de um grafo (3) cálculo de densidade do grafo. Respectivamente, cada etapa concretiza uma contribuição, em particular (1) um filtro desenhado para ser aplicado nos dados EEG para utilização em tarefas de compreensão de código; (2) definição de propriedades para atribuir pesos e relevância em relação a interação entre os canais; (3) quantificação da conectividade através do uso da densidade ponderada do grafo.

(3) Evidência da correlação da abordagem CogEff e abordagens tradicionais baseadas em EEG com a compreensão de código: esta tese demonstrou uma avaliação que apresenta uma correlação da abordagem CogEff e abordagens tradicionais baseadas em EEG a compreensão de código. Antes, existiam apenas avaliações superficiais e pontuais de isoladas abordagens em relação a propósitos relacionados a compreensão de código, tais como nível de dificuldade e nível de experiência. Especificamente essa evidência de correlação também demonstrou (1) como correlacionar abordagens para analisar carga cognitiva com tarefas de compreensão de código. Por exemplo, correlação de abordagens EEG com aspectos psicométricos da carga cognitiva tais como o tempo, e os dados categóricos das respostas de tarefas experimentais; (2) uma primeira evidência da correlação da abordagem CogEff com tarefas de compreensão de código; (3) a diferença entre tarefas de complexidade alta e complexidade baixa pela CogEff. No geral, os testes apresentaram a correlação da CogEff com as tarefas de compreensão de código do *dataset*.

(4) Classificação de compreensão de código por técnicas de *machine learning* baseados em dados EEG de desenvolvedores de software com alta efetividade: Através da métrica de *f-measure*, foi evidenciado que técnicas de *machine learning K-Nearest Neighbor*, *Neural Network* e *Random Forest* podem classificar compreensão de código baseado em dados EEG com efetividade muito alta ($\geq$ 80%). Além disso, essa classificação contribuiu com (1) um

método para tratar dados de EEG e prepará-los para treinamento em técnicas de *machine learning*; (2) como avaliar a efetividade de técnicas de *machine learning* treinadas com dados EEG para classificar compreensão de código, inclusive, de como comparar a efetividade dessas técnicas são significativamente maiores que classificadores randômicos; (3) lista de implicações de classificar compreensão de código baseado em dados de EEG na indústria e academia.

7.3 Trabalhos Futuros

Esta seção destaca os principais trabalhos futuros em relação a essa tese. Os trabalhos futuros foram derivados ao longo da tese com objetivo de principalmente prestar continuidade as contribuições que foram concretizadas nesta tese. Essas contribuições, portanto, foram um primeiro passo para investigar a relação das abordagens EEG diretamente com a compreensão de código. Os trabalhos futuros são apresentados primeiramente em relação a cada questão de pesquisa respondida.

Ao responder à questão de pesquisa 1 (QP1) foi disponibilizado uma classificação do estado da arte sobre mensurar a carga cognitiva na área de engenharia de software. Em relação ao mapeamento sistemático da literatura sobre as medidas de carga cognitiva aplicas na área de engenharia de software, será fundamental em estudos futuros atualizar periodicamente a relação de estudos primários selecionados. O mapeamento sistemático da literatura presente nesta tese decorre sobre estudos publicados até março de 2020. É fundamental que trabalhos futuros atualizem essa lista, através de uma pesquisa de replicação, isto é, replicando os mesmos protocolos do mapeamento realizado nesta tese. É importante também atualizar esses estudos implicando também, na atualização dos resultados em relação aos aspectos investigados, isto é, as definições de carga cognitiva, os dispositivos utilizados, as métricas, as técnicas de *machine learning*, os propósitos de utilização da carga cognitiva, as tarefas, artefatos, participantes, métodos de pesquisas, e locais de publicação. Isto implicará e possibilitará futuramente derivar novas tendências e desafios de pesquisas atuais. Futuramente, em relação a esse capítulo, também poderá ser produzido uma revisão sistemática da literatura sobre carga cognitiva na área de engenharia de software. Revisões sistemáticas tem a característica de analisar o assunto com mais profundidade, em particular, esse estudo poderá focar na área de compreensão de código, e comparar o estado arte das abordagens de forma analítica.

Após investigar a questão de pesquisa 2 (QP2) foi disponibilizado uma abordagem baseada em EEG para mensurar a carga cognitiva em tarefas de compreensão de código. A abordagem CogEff (*Cognitive Effort*), que se baseia em dados EEG para mensurar a carga cognitiva dos desenvolvedores em tarefas de compreensão de código, foi concebida com o propósito de possibilitar extensões em relação a cada passo dessa abordagem em trabalhos futuros. Por exemplo, trabalhos podem futuramente propor outros tipos de filtros de EEG para serem utilizados na abordagem CogEff. Para isso, trabalhos futuros podem replicar a metodologia de GONÇALES et al. (2021), para testar a eficácia de um novo filtro para sinais de EEG. Além disso, futuros

trabalhos poderão estender o modo de utilização das propriedades definidas para a CogEff. As propriedades definidas neste trabalho têm o objetivo de fornecer uma forma inicial de penalizar as relações existentes entre canais, para atribuir relevância a aquelas interações, que, de acordo com literatura, tem mais relevância com a compreensão de código. Neste trabalho, o critério utilizado foram pesos atribuídos em relação as áreas cerebrais que foram ativadas durante tarefas de compreensão de código em pesquisas utilizando fMRI. Futuros trabalhos poderiam fornecer parametrizações recomendadas para a CogEff, através da utilização de algoritmos de *grid* e de otimização para encontrar os melhores parâmetros possíveis para a abordagem. Além disso, avaliações sobre a efetividade entre os uma abordagem CogEff sem parâmetros em relação aos resultados a uma abordagem que definem esses parâmetros podem ser conduzidos. Além disso, o cálculo de densidade também pode ser estendido em estudos futuros. A CogEff futuramente também pode se tornar em um projeto *open source*, com melhorias e avanços implementados de forma incremental e disponibilizadas pela comunidade acadêmica.

Ao responder à questão de pesquisa (QP3) foi fornecida uma análise da correlação entre a CogEff e tarefas de compreensão de código. Além disso, a correlação das abordagens tradicionais de EEG também foram avaliadas em relação a compreensão de código. Em relação a essa questão, trabalhos futuros poderão estender a análise para um *dataset* maior. Nesta tese, a avaliação da correlação foi feita em relação a uma base de dados contendo dados EEG de 35 participantes, que realizaram 10 tarefas de compreensão de código. Esse é um fator que limita a generalização dos resultados da análise de correlação da CogEff e das demais abordagens EEG. Para isso, trabalhos futuros poderiam reproduzir a análise de correlação em um *dataset* que contenha um número representativo de desenvolvedores, isto é, que contenha uma quantidade mais diversificada de perfis. Ao mesmo tempo, trabalhos futuros também poderiam estender a lista de tarefas de compreensão, pois as tarefas utilizadas se limitaram a tarefas elaboradas em linguagem Java, e com tamanho de no máximo 44 linhas, e uma complexidade ciclomática média de 3,4. Seria interessante coletar dados relacionados a carga cognitiva em tarefas em outras linguagens de programação e com complexidade maior. Além disso, a análise de correlação poderia ser estendida de modo a incluir outros tipos de abordagens de EEG utilizadas para mensurar a carga cognitiva, tais como, as variadas taxas de frequência de bandas do EEG. Se todas essas variações fossem consideradas, o estudo abrangeria um escopo além da avaliação das principais abordagens EEG. Futuros estudos também poderiam avaliar a correlação da CogEff em relação a compreensão de código em diferentes cenários de parametrizações. Por fim, a diferença da CogEff entre tarefas de complexidade maior e de complexidade menor, poderia ser testada e estendida com base em um *dataset* maior e fornecer mais resultados para reforçar a generalização da abordagem proposta.

Após investigar a questão de pesquisa 4 (QP4), uma classificação da compreensão de código utilizando técnicas de *machine learning* treinadas com EEG foi realizada. Durante essa investigação, uma análise de efetividade entre as diferentes técnicas foi realizada. Em relação a isso, trabalhos futuros poderiam utilizar o *dataset* estendido sugerido no parágrafo anterior,

para replicar o estudo realizado com o objetivo de treinar as mesmas técnicas para classificar compreensão de código e realizar uma nova análise, isto é, o quanto os novos dados impactaram na efetividade dos classificadores. Além disso, estudos futuros também poderiam adotar o *esamble machine learning*, o qual não foi avaliado neste trabalho. O *esamble learning* consiste na combinação de várias técnicas de *machine learning*, onde os resultados de cada técnica, pode ser combinado por um sistema de ranking ou votação. O *esamble machine learning* tem potencial de extrair resultados melhores por combinar várias técnicas de *machine learning* com características diferentes. Por fim, em relação a investigação da questão de pesquisa 4, os cenários de classificação podem ser estendidos. Por exemplo, a classificação realizada neste trabalho consiste em uma validação cruzada de todo o *dataset*. Porém, trabalhos futuros podem ir além disso, e avaliar a efetividade da classificação nos casos em que se classifica novas tarefas, isto é, quando um desenvolvedor conhecido pelo *dataset* realiza tarefas de compreensão que ainda não são conhecidas pelo classificador. Outro cenário seria testar a efetividade das técnicas em um cenário que classifica compreensão de código de novos desenvolvedores, isto é, desenvolvedores que estão realizando tarefas conhecidas pelo classificador, mas com dados de EEG de um desenvolvedor desconhecido pelos classificadores. Por fim a efetividade das técnicas de *machine learning* para classificar compreensão de código poderia ser avaliada considerando os dois casos, isto é, uma classificação de compreensão de código baseado nos dados de um novo desenvolvedor conduzindo uma nova tarefa de compreensão.

Por fim, um possível trabalho futuro, além desta tese, seria a condução de um experimento controlado para analisar a diferença de esforço mental entre tarefas de compreensão de código que são legíveis e tarefas de compreensão de código que são ilegíveis. Uma pesquisa deste tipo foi realizada por FAKHOURY et al. (2018) mas utilizando fNRIS ao invés de EEG. PEITEK et al. (2018b) mediu a ativação de áreas cerebrais durante de desenvolvedores durante tarefas de compreensão de código que possuíam identificadores em língua alemã e não indentado, em relação a ativação cerebral durante a condução de tarefas de compreensão em tarefas com um código fonte em língua inglesa e indentado, porém, o estudo foi realizado baseado em dados do fMRI, e não com dados de EEG. Enquanto FAKHOURY et al. (2018) evidenciou diferença na carga cognitiva aplicada em tarefas com qualidade código prejudicada, PEITEK et al. (2018b) não encontrou essa diferença. Por isso, novos estudos seriam necessários para explorar essa divergência, e, com possibilidade da utilização de dados de EEG. Além de uma possível contribuição em relação ao impacto da qualidade e legibilidade do código fonte na carga cognitiva, também seria uma oportunidade de utilizar com mais frequência as abordagens de EEG para medir a carga cognitiva nos estudos da área de engenharia de software.

REFERÊNCIAS

ANTONENKO, P.; PAAS, F.; GRABNER, R.; VAN GOG, T. Using electroencephalography to measure cognitive load. **Educational psychology review**, [S.l.], v. 22, n. 4, p. 425–438, 2010.

BABLANI, A.; EDLA, D. R.; TRIPATHI, D.; CHERUKU, R. Survey on brain-computer interface: an emerging computational intelligence paradigm. **ACM Comput. Surv.**, New York, NY, USA, v. 52, n. 1, p. 20:1–20:32, Feb. 2019.

BATESON, A. D.; BASELER, H. A.; PAULSON, K. S.; AHMED, F.; ASGHAR, A. U. Categorisation of mobile EEG: a researcher's perspective. **BioMed research international**, [S.l.], v. 2017, 2017.

BEHUTIYE, W.; KARHAPää, P.; LóPEZ, L.; BURGUéS, X.; MARTíNEZ-FERNáNDEZ, S.; VOLLMER, A. M.; RODRíGUEZ, P.; FRANCH, X.; OIVO, M. Management of quality requirements in agile and rapid software development: a systematic mapping study. **Information and Software Technology**, [S.l.], v. 123, p. 106225, 2020.

BENESTY, J.; CHEN, J.; HUANG, Y.; COHEN, I. Pearson correlation coefficient. In: **Noise reduction in speech processing**. [S.l.]: Springer, 2009. p. 1–4.

BENGIO, Y.; GRANDVALET, Y. No unbiased estimator of the variance of k-fold cross-validation. **Journal of machine learning research**, [S.l.], v. 5, n. Sep, p. 1089–1105, 2004.

BISCHOFF, V.; FARIAS, K.; GONÇALES, L. J.; BARBOSA, J. L. V. Integration of feature models: a systematic mapping study. **Information and Software Technology**, [S.l.], v. 105, p. 209–225, 2019.

BLASCO, J.; CHEN, T. M.; TAPIADOR, J.; PERIS-LOPEZ, P. A survey of wearable biometric recognition systems. **ACM Computing Surveys**, New York, NY, USA, v. 49, n. 3, p. 43:1–43:35, Sept. 2016.

BOUGHORBEL, S.; JARRAY, F.; EL-ANBARI, M. Optimal classifier for imbalanced data using matthews correlation coefficient metric. **PloS one**, [S.l.], v. 12, n. 6, p. e0177678, 2017.

BRAISBY, N.; GELLATLY, A. **Cognitive psychology**. [S.l.]: Oxford University Press, 2012.

CALDIERA, V. R. B. G.; ROMBACH, H. D. The goal question metric approach. **Encyclopedia of software engineering**, [S.l.], p. 528–532, 1994.

CASTELHANO, J.; DUARTE, I. C.; FERREIRA, C.; DURAES, J.; MADEIRA, H.; CASTELO-BRANCO, M. The role of the insula in intuitive expert bug detection in computer code: an fmri study. **Brain imaging and behavior**, [S.l.], v. 13, n. 3, p. 623–637, 2019.

CHEVEIGNÉ, A. de; NELKEN, I. Filters: when, why, and how (not) to use them. **Neuron**, [S.l.], v. 102, n. 2, p. 280–293, 2019.

COHEN, M. X. **Analyzing neural time series data**: theory and practice. [S.l.]: MIT press, 2014.

COHEN, M. X. Where does eeg come from and what does it mean? **Trends in neurosciences**, [S.l.], v. 40, n. 4, p. 208–218, 2017.

COOLEY, J. W.; TUKEY, J. W. An algorithm for the machine calculation of complex fourier series. **Mathematics of computation**, [S.l.], v. 19, n. 90, p. 297–301, 1965.

CRK, I.; KLUTHE, T. Assessing the contribution of the individual alpha frequency (IAF) in an EEG-based study of program comprehension. In: ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY (EMBC), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. p. 4601–4604.

CRK, I.; KLUTHE, T.; STEFIK, A. Understanding programming expertise: an empirical study of phasic brain wave changes. **ACM Trans. on Computing-Human Interaction**, New York, NY, USA, v. 23, n. 1, p. 2:1–2:29, dec 2015.

DAVIES, K. S. Formulating the evidence based practice question: a review of the frameworks. **Evidence Based Library and Information Practice**, [S.l.], v. 6, n. 2, p. 75–80, 2011.

DEBENER, S.; ULLSPERGER, M.; SIEGEL, M.; FIEHLER, K.; VON CRAMON, D. Y.; ENGEL, A. K. Trial-by-trial coupling of concurrent electroencephalogram and functional magnetic resonance imaging identifies the dynamics of performance monitoring. **Journal of Neuroscience**, [S.l.], v. 25, n. 50, p. 11730–11737, 2005.

DEEPA, N.; PHAM, Q.-V.; NGUYEN, D. C.; BHATTACHARYA, S.; PRABADEVI, B.; GADEKALLU, T. R.; MADDIKUNTA, P. K. R.; FANG, F.; PATHIRANA, P. N. A survey on blockchain for big data: approaches, opportunities, and future directions. **Future Generation Computer Systems**, [S.l.], 2022.

DEVORE, J. L.; FARNUM, N. R.; DOI, J. A. **Applied statistics for engineers and scientists**. [S.l.]: Cengage Learning, 2013.

DIRICAN, A. C.; GÖKTÜRK, M. Psychophysiological measures of human cognitive states applied in human computer interaction. **Procedia Computer Science**, [S.l.], v. 3, p. 1361–1367, 2011.

DURAISINGAM, A.; PALANIAPPAN, R.; ANDREWS, S. Cognitive task difficulty analysis using eeg and data mining. In: CONFERENCE ON EMERGING DEVICES AND SMART SYSTEMS (ICEDSS), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 52–57.

EIBE, F.; HALL, M. A.; WITTEN, I. H. The weka workbench. online appendix for data mining: practical machine learning tools and techniques. In: **Morgan kaufmann**. [S.l.]: Elsevier Amsterdam, The Netherlands, 2016.

EMOTIV. **Testbench™ specifications, emotiv, 2014**. 2014.

Emotiv Systems. **Emotiv EPOC**. 2021.

FAIRHURST, M.; LI, C.; COSTA-ABREU, M. D. Predictive biometrics: a review and analysis of predicting personal characteristics from biometric data. **IET Biometrics**, [S.l.], v. 6, n. 6, p. 369–378, 2017.

FAKHOURY, S.; MA, Y.; ARNAOUDOVA, V.; ADESOPE, O. The effect of poor source code lexicon and readability on developers' cognitive load. In: INT'L CONF. PROGRAM COMPREHENSION (ICPC), 2018. **Proceedings...** [S.l.: s.n.], 2018.

FARIAS, K.; GARCIA, A.; LUCENA, C. Effects of stability on model composition effort: an exploratory study. **Software & Systems Modeling**, [S.l.], v. 13, n. 4, p. 1473–1494, 2014.

FARIAS, K.; GARCIA, A.; WHITTLE, J.; CHAVEZ, C. v. F. G.; LUCENA, C. Evaluating the effort of composing design models: a controlled experiment. **Software & Systems Modeling**, [S.l.], v. 14, n. 4, p. 1349–1365, 2015.

FIGL, K. Comprehension of procedural visual business process models. **Business & Information Systems Engineering**, [S.l.], v. 59, n. 1, p. 41–67, Feb 2017.

FLOYD, B.; SANTANDER, T.; WEIMER, W. Decoding the representation of code in the brain: an fmri study of code review and expertise. In: IEEE/ACM 39TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 175–186.

FRITZ, T.; BEGEL, A.; MüLLER, S. C.; YIGIT-ELLIOTT, S.; ZüGER, M. Using psycho-physiological measures to assess task difficulty in software development. In: INT. CONFERENCE ON SOFTWARE ENGINEERING, 36., 2014. **Proceedings...** [S.l.: s.n.], 2014. p. 402–413. (ICSE 2014).

FRITZ, T.; MÜLLER, S. C. Leveraging biometric data to boost software developer productivity. In: IEEE 23RD INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION, AND REENGINEERING (SANER), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. v. 5, p. 66–77.

FUCCI, D.; GIRARDI, D.; NOVIELLI, N.; QUARANTA, L.; LANUBILE, F. A replication study on code comprehension and expertise using lightweight biometric sensors. In: IEEE/ACM 27TH INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION (ICPC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p. 311–322.

GONÇALES, L.; FARIAS, K. Towards the measurement of mental effort in software engineering: a research agenda. **International Journal of Computer Applications**, [S.l.], v. 975, p. 8887, 2020.

GONCALES, L.; FARIAS, K. **CogEff - uma abordagem baseada em EEG para mensurar carga cognitiva de desenvolvedores em tarefas de compreensão de código - Código Fonte**. Google Drive, `https://drive.google.com/drive/folders/1EpViP11CGqI-nWg7CILgFR13QnDM3fIr?usp=sharing`.

GONCALES, L.; FARIAS, K.; KUPSSISNKÜ, L.; SEGALOTTO, M. **An empirical evaluation of machine learning techniques to classify code comprehension based on EEG data**: supplementary material. Mendeley Data,V1, `https://doi.org/10.17632/g4pvnwt4jc.1`.

GONCALES, L.; FARIAS, K.; SILVA, B. d.; FESSLER, J. Measuring the cognitive load of software developers: a systematic mapping study. In: INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION, 27., 2019, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2019. p. 42–52. (ICPC '19).

GONÇALES, L. J.; FARIAS, K.; KUPSSINSKÜ, L. S.; SEGALOTTO, M. Evaluation of machine learning techniques to classify code comprehension based on developers' eeg data. In: BRAZILIAN SYMPOSIUM ON HUMAN FACTORS IN COMPUTING SYSTEMS, 19., 2020. **Proceedings...** [S.l.: s.n.], 2020. p. 1–10.

GONÇALES, L. J.; FARIAS, K.; KUPSSINSKÜ, L.; SEGALOTTO, M. The effects of applying filters on eeg signals for classifying developers' code comprehension. **Journal of Applied Research and Technology**, [S.l.], v. 19, n. 6, p. 584–602, 2021.

GONÇALES, L. J.; FARIAS, K.; OLIVEIRA, T. C. D.; SCHOLL, M. Comparison of software design models: an extended systematic mapping study. **ACM Computing Surveys (CSUR)**, [S.l.], v. 52, n. 3, p. 1–41, 2019.

GONCALES, L. J.; FARIAS, K.; SILVA, B. C. da. Measuring the cognitive load of software developers: an extended systematic mapping study. **Information and Software Technology**, [S.l.], v. 136, p. 106563, 2021.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A.; BENGIO, Y. **Deep learning**. [S.l.]: MIT press Cambridge, 2016. v. 1, n. 2.

GUI, Q.; RUIZ-BLONDET, M. V.; LASZLO, S.; JIN, Z. A survey on brain biometrics. **ACM Comput. Surv.**, New York, NY, USA, v. 51, n. 6, p. 112:1–112:38, feb 2019.

HALL, M.; FRANK, E.; HOLMES, G.; PFAHRINGER, B.; REUTEMANN, P.; WITTEN, I. H. The weka data mining software: an update. **ACM SIGKDD explorations newsletter**, [S.l.], v. 11, n. 1, p. 10–18, 2009.

HARRIS, F. J. On the use of windows for harmonic analysis with the discrete fourier transform. **Proceedings of the IEEE**, [S.l.], v. 66, n. 1, p. 51–83, 1978.

HIJAZI, H.; DURAES, J.; COUCEIRO, R.; CASTELHANO, J.; BARBOSA, R.; MEDEIROS, J.; CASTELO-BRANCO, M.; DE CARVALHO, P.; MADEIRA, H. Quality evaluation of modern code reviews through intelligent biometric program comprehension. **IEEE Transactions on Software Engineering**, [S.l.], 2022.

HUANG, X.; LIN, J.; DEMNER-FUSHMAN, D. Evaluation of pico as a knowledge representation for clinical questions. In: AMIA ANNUAL SYMPOSIUM PROCEEDINGS, 2006. **Anais...** [S.l.: s.n.], 2006. v. 2006, p. 359.

HUANG, Y.; LIU, X.; KRUEGER, R.; SANTANDER, T.; HU, X.; LEACH, K.; WEIMER, W. Distilling neural representations of data structure manipulation using fmri and fnirs. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 41., 2019. **Proceedings...** [S.l.: s.n.], 2019. p. 396–407.

HYVARINEN, A. Fast and robust fixed-point algorithms for independent component analysis. **IEEE transactions on Neural Networks**, [S.l.], v. 10, n. 3, p. 626–634, 1999.

HYVÄRINEN, A.; OJA, E. Independent component analysis: algorithms and applications. **Neural networks**, [S.l.], v. 13, n. 4-5, p. 411–430, 2000.

KARAS, Z.; JAHN, A.; WEIMER, W.; HUANG, Y. Connecting the dots: rethinking the relationship between code and prose writing with functional connectivity. In: ACM JOINT MEETING ON EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING, 29., 2021. **Proceedings...** [S.l.: s.n.], 2021. p. 767–779.

KATONA, J.; FARKAS, I.; UJBANYI, T.; DUKAN, P.; KOVARI, A. Evaluation of the neurosky mindflex eeg headset brain waves data. In: IEEE 12TH INTERNATIONAL SYMPOSIUM ON APPLIED MACHINE INTELLIGENCE AND INFORMATICS (SAMI), 2014., 2014. **Anais...** [S.l.: s.n.], 2014. p. 91–94.

KENDALL, M. G. A new measure of rank correlation. **Biometrika**, [S.l.], v. 30, n. 1-2, p. 81–93, 1938.

KHANAGHA, S.; VOLBERDA, H. W.; ALEXIOU, A.; ANNOSI, M. C. Mitigating the dark side of agile teams: peer pressure, leaders' control, and the innovative output of agile teams. **Journal of Product Innovation Management**, [S.l.], 2021.

KITCHENHAM, B. A.; BUDGEN, D.; PEARL BRERETON, O. Using mapping studies as the basis for further research - a participant-observer case study. **Information Software Technology**, Newton, MA, USA, v. 53, n. 6, p. 638–651, June 2011.

KITCHENHAM, B. A.; MENDES, E.; TRAVASSOS, G. H. Cross versus within-company cost estimation studies: a systematic review. **IEEE Trans. on Software Engineering**, [S.l.], v. 33, n. 5, p. 316–329, May 2007.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing systematic literature reviews in software engineering**. 2007.

KLIMESCH, W. EEG alpha and theta oscillations reflect cognitive and memory performance: a review and analysis. **Brain research reviews**, [S.l.], v. 29, n. 2-3, p. 169–195, 1999.

KOSTI, M. V.; GEORGIADIS, K.; ADAMOS, D. A.; LASKARIS, N.; SPINELLIS, D.; ANGELIS, L. Towards an affordable brain computer interface for the assessment of programmers' mental workload. **International Journal of Human-Computer Studies**, [S.l.], v. 115, p. 52–66, 2018.

LACHAUX, J.-P.; AXMACHER, N.; MORMANN, F.; HALGREN, E.; CRONE, N. E. High-frequency neural activity and human cognition: past, present and possible future of intracranial eeg research. **Progress in Neurobiology**, [S.l.], v. 98, n. 3, p. 279–301, 2012. High Frequency Oscillations in Cognition and Epilepsy.

LEE, S.; HOOSHYAR, D.; JI, H.; NAM, K.; LIM, H. Mining biometric data to predict programmer expertise and task difficulty. **Cluster Computing**, [S.l.], Jan 2017.

LIU, G.; WONG, L.; CHUA, H. N. Complex discovery from weighted PPI networks. **Bioinformatics**, [S.l.], v. 25, n. 15, p. 1891–1897, 05 2009.

LOTTE, F.; BOUGRAIN, L.; CICHOCKI, A.; CLERC, M.; CONGEDO, M.; RAKOTOMAMONJY, A.; YGER, F. A review of classification algorithms for eeg-based brain–computer interfaces: a 10 year update. **Journal of neural engineering**, [S.l.], v. 15, n. 3, p. 031005, 2018.

LOTTE, F.; CONGEDO, M.; LÉCUYER, A.; LAMARCHE, F.; ARNALDI, B. A review of classification algorithms for EEG-based brain–computer interfaces. **Journal of Neural Engineering**, [S.l.], v. 4, n. 2, p. R1–R13, jan 2007.

LUND, H. Eye tracking in library and information science: a literature review. **Library Hi Tech**, [S.l.], v. 34, n. 4, p. 585–614, 2016.

MAIORANA, E.; SOLÉ-CASALS, J.; CAMPISI, P. Eeg signal preprocessing for biometric recognition. **Machine Vision and Applications**, [S.l.], v. 27, n. 8, p. 1351–1360, 2016.

MEYER, A. N.; ZIMMERMANN, T.; FRITZ, T. Characterizing software developers by perceptions of productivity. In: ACM/IEEE INTERNATIONAL SYMPOSIUM ON EMPIRICAL SOFTWARE ENGINEERING AND MEASUREMENT, 11., 2017. **Proceedings...** [S.l.: s.n.], 2017. p. 105–110.

MICHEL, C. M.; HE, B. Eeg source localization. **Handbook of clinical neurology**, [S.l.], v. 160, p. 85–101, 2019.

MOHANANI, R.; SALMAN, I.; TURHAN, B.; RODRíGUEZ, P.; RALPH, P. Cognitive biases in software engineering: a systematic mapping study. **IEEE Transactions on Software Engineering**, [S.l.], 2018.

MULERT, C.; LEMIEUX, L. **Eeg-fmri**: physiological basis, technique, and applications. [S.l.]: Springer Science & Business Media, 2009.

MÜLLER, S. C. Measuring software developers' perceived difficulty with biometric sensors. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - VOLUME 2, 37., 2015, Piscataway, NJ, USA. **Proceedings...** IEEE Press, 2015. p. 887–890. (ICSE '15).

MÜLLER, S. C.; FRITZ, T. Stuck and frustrated or in flow and happy: sensing developers' emotions and progress. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - VOLUME 1, 37., 2015. **Proceedings...** IEEE Press, 2015. p. 688–699. (ICSE '15).

MÜLLER, S. C.; FRITZ, T. Using (bio)metrics to predict code quality online. In: INT. CONF. ON SOFTWARE ENGINEERING, 38., 2016, New York, NY, USA. **Proceedings...** ACM, 2016. p. 452–463. (ICSE '16).

OBAIDELLAH, U.; AL HAEK, M.; CHENG, P. C.-H. A survey on the usage of eye-tracking in computer programming. **ACM Computing Surveys (CSUR)**, [S.l.], v. 51, n. 1, p. 1–58, 2018.

PAAS, F. G.; VAN MERRIËNBOER, J. J. Instructional control of cognitive load in the training of complex cognitive tasks. **Educational psychology review**, [S.l.], v. 6, n. 4, p. 351–371, 1994.

PALANIAPPAN, R. **Biological signal analysis**. [S.l.]: BookBoon, 2011.

Paradigm Software. **Paradigm**. 2019.

PARKS, T. W.; BURRUS, C. S. **Digital filter design**. [S.l.]: Wiley-Interscience, 1987.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: machine learning in Python. **Journal of Machine Learning Research**, [S.l.], v. 12, p. 2825–2830, 2011.

PEITEK, N.; APEL, S.; BRECHMANN, A.; PARNIN, C.; SIEGMUND, J. Codersmuse: multi-modal data exploration of program-comprehension experiments. In: INTERNATIONAL CONFERENCE ON PROGRAM COMPREHENSION, 27., 2019. **Proceedings...** [S.l.: s.n.], 2019. p. 126–129.

PEITEK, N.; APEL, S.; PARNIN, C.; BRECHMANN, A.; SIEGMUND, J. Program comprehension and code complexity metrics: an fmri study. In: IEEE/ACM 43RD INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2021., 2021. **Anais...** [S.l.: s.n.], 2021. p. 524–536.

PEITEK, N.; SIEGMUND, J.; APEL, S.; KäSTNER, C.; PARNIN, C.; BETHMANN, A.; LEICH, T.; SAAKE, G.; BRECHMANN, A. A look into programmers' heads. **IEEE Transactions on Software Engineering**, [S.l.], p. 1–20, 2018.

PEITEK, N.; SIEGMUND, J.; PARNIN, C.; APEL, S.; BRECHMANN, A. Toward conjoint analysis of simultaneous eye-tracking and fmri data for program-comprehension studies. In: WORKSHOP ON EYE MOVEMENTS IN PROGRAMMING, 2018, New York, NY, USA. **Proceedings...** ACM, 2018. p. 1:1–1:5. (EMIP '18).

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: an update. **Information and Software Technology**, [S.l.], v. 64, p. 1–18, 2015.

PLASS, J. L.; KALYUGA, S. Four ways of considering emotion in cognitive load theory. **Educational Psychology Review**, [S.l.], p. 1–21, 2019.

QIU, D.; LI, B.; JI, S.; LEUNG, H. Regression testing of web service: a systematic mapping study. **ACM Computing Surveys**, New York, NY, USA, v. 47, n. 2, p. 21:1–21:46, aug 2014.

RADEVSKI, S.; HATA, H.; MATSUMOTO, K. Real-time monitoring of neural state in assessing and improving software developers' productivity. In: INTERNATIONAL WORKSHOP ON COOPERATIVE AND HUMAN ASPECTS OF SOFTWARE ENGINEERING, 2015, Piscataway, NJ, USA. **Anais...** [S.l.: s.n.], 2015. p. 93–96. (CHASE '15).

RISH, I. An empirical study of the naive bayes classifier. In: IJCAI 2001 WORKSHOP ON EMPIRICAL METHODS IN ARTIFICIAL INTELLIGENCE, 2001. **Anais...** [S.l.: s.n.], 2001. v. 3, n. 22, p. 41–46.

SCALABRINO, S.; BAVOTA, G.; VENDOME, C.; LINARES-VÁSQUEZ, M.; POSHYVANYK, D.; OLIVETO, R. Automatically assessing code understandability: how far are we? In: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 32., 2017. **Proceedings...** [S.l.: s.n.], 2017. p. 417–427.

SCALABRINO, S.; BAVOTA, G.; VENDOME, C.; POSHYVANYK, D.; OLIVETO, R. et al. Automatically assessing code understandability. **IEEE Transactions on Software Engineering**, [S.l.], 2019.

SEGALOTTO, M. **Arni an eeg based model to measure program comprehension**. 2018. Tese (Doutorado em Ciência da Computação) — Universidade do Vale do Rio dos Sinos, 2018.

SHARAFI, Z.; SOH, Z.; GUÉHÉNEUC, Y.-G. A systematic literature review on the usage of eye-tracking in software engineering. **Information and Software Technology**, [S.l.], v. 67, p. 79–107, 2015.

SIEGMUND, J. Program comprehension: past, present, and future. In: IEEE 23RD INTERNATIONAL CONFERENCE ON SOFTWARE ANALYSIS, EVOLUTION, AND REENGINEERING (SANER), 2016., 2016. **Anais...** [S.l.: s.n.], 2016. v. 5, p. 13–20.

SIEGMUND, J.; KÄSTNER, C.; APEL, S.; PARNIN, C.; BETHMANN, A.; LEICH, T.; SAAKE, G.; BRECHMANN, A. Understanding understanding source code with functional magnetic resonance imaging. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 36., 2014. **Proceedings...** [S.l.: s.n.], 2014. p. 378–389.

SIEGMUND, J.; PEITEK, N.; PARNIN, C.; APEL, S.; HOFMEISTER, J.; KäSTNER, C.; BEGEL, A.; BETHMANN, A.; BRECHMANN, A. Measuring neural efficiency of program comprehension. In: JOINT MEETING ON FOUNDATIONS OF SOFTWARE ENGINEERING, 2017., 2017. **Proceedings...** ACM, 2017. p. 140–150. (ESEC/FSE 2017).

SIEGMUND, J.; SIEGMUND, N.; APEL, S. Views on internal and external validity in empirical software engineering. In: IEEE INT. CONFERENCE ON SOFTWARE ENGINEERING, 37., 2015. **Anais...** [S.l.: s.n.], 2015. v. 1, p. 9–19.

SOLOWAY, E.; EHRLICH, K. Empirical studies of programming knowledge. **IEEE Transactions on software engineering**, [S.l.], n. 5, p. 595–609, 1984.

SONARQUBE. **Sonar Qube 8.9 lts**. `https://www.sonarqube.org/`, Accessed on June 2021.

SPEARMAN, C. The proof and measurement of association between two things. **The American journal of psychology**, [S.l.], v. 100, n. 3-4, p. 441–471, 1987.

STIPACEK, A.; GRABNER, R.; NEUPER, C.; FINK, A.; NEUBAUER, A. Sensitivity of human eeg alpha band desynchronization to different working memory components and increasing levels of memory load. **Neuroscience letters**, [S.l.], v. 353, n. 3, p. 193–196, 2003.

STOL, K.-J.; RALPH, P.; FITZGERALD, B. Grounded theory in software engineering research: a critical review and guidelines. In: INT. CONFERENCE ON SOFTWARE ENGINEERING, 38., 2016. **Proceedings...** [S.l.: s.n.], 2016. p. 120–131.

SWELLER, J. Element interactivity and intrinsic, extraneous, and germane cognitive load. **Educational Psychology Review**, [S.l.], v. 22, n. 2, p. 123–138, Jun 2010.

THARWAT, A. Classification assessment methods. **Applied Computing and Informatics**, [S.l.], 2020.

TROCKMAN, A.; CATES, K.; MOZINA, M.; NGUYEN, T.; KÄSTNER, C.; VASILESCU, B. Automatically assessing code understandability reanalyzed: combined metrics matter. In: INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES, 15., 2018. **Proceedings...** [S.l.: s.n.], 2018. p. 314–318.

UNO, T. An efficient algorithm for enumerating pseudo cliques. In: INTERNATIONAL SYMPOSIUM ON ALGORITHMS AND COMPUTATION, 2007. **Anais...** [S.l.: s.n.], 2007. p. 402–414.

VIDAL, S. A.; MARCOS, C.; DÍAZ-PACE, J. A. An approach to prioritize code smells for refactoring. **Automated Software Engineering**, [S.l.], v. 23, n. 3, p. 501–532, 2016.

WIERINGA, R.; MAIDEN, N.; MEAD, N.; ROLLAND, C. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. **Requirements Engineering**, Secaucus, NJ, USA, v. 11, n. 1, p. 102–107, december 2005.

WILBANKS, B. A.; MCMULLAN, S. P. A review of measuring the cognitive workload of electronic health records. **CIN: Computers, Informatics, Nursing**, [S.l.], v. 36, n. 12, p. 579–588, 2018.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. **Experimentation in software engineering**. [S.l.]: Springer Science & Business Media, 2012.

YEH, M. K.-C.; GOPSTEIN, D.; YAN, Y.; ZHUANG, Y. Detecting and comparing brain activity in short program comprehension using eeg. In: IEEE FRONTIERS IN EDUCATION CONFERENCE (FIE), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 1–5.

ZIMOCH, M.; PRYSS, R.; SCHOBEL, J.; REICHERT, M. Eye tracking experiments on process model comprehension: lessons learned. In: **Enterprise, business-process and information systems modeling**. [S.l.]: Springer, 2017. p. 153–168.

ZÜGER, M.; FRITZ, T. Sensing and supporting software developers' focus. In: CONFERENCE ON PROGRAM COMPREHENSION, 26., 2018. **Proceedings...** [S.l.: s.n.], 2018. p. 2–6.

ANEXO A – TAREFAS DE COMPREENSÃO

Este anexo apresenta as tarefas de compreensão de código SEGALOTTO (2018), que foram utilizadas no dataset utilizado neste trabalho.

Figura 29: Tarefa T1 - Processar Salário.

```
1. public class Payment {
2.      public static void main(String[] args) {
3.             double bill = 15;
4.             bill = bill - 5;
5.             bill = bill * 2;
6.             bill = bill + 10 ;
7.             bill = bill / 5 ;
8.             System.out.print( ++bill );
9.      }
10. }
```

(1) 5 (2) 7 - Correta (3) 3 (4) 6 (5) 14

Fonte: SEGALOTTO (2018)

Figura 30: Tarefa T2 - Processar Salário v2.

```
1. public class PaymentV2 {
2.     public static void main(String[] args) {
3.         double bill = 13;
4.         payIfNecessary ( bill - 7 );
5.     }
6.     public static void payIfNecessary ( double amount) {
7.         calculateAndDeliverPay ( amount * 2 );
8.     }
9.     private static void calculateAndDeliverPay ( double pay ) {
10.        deliveryPay ( pay + 15 );
11.    }
12.    private static void deliveryPay ( double value ) {
13.        calculateValue ( value / 3 );
14.    }
15.    private static void calculateValue ( double total ) {
16.        System.out.print( ++total );
17.    }
18. }
```

(1) 11 (2) 13 (3) 12 (4) 10 - Correta (5) 9

Fonte: SEGALOTTO (2018)

Figura 31: Tarefa T3 - Processar Pontos.

```
1. public class Soccer {
2.     public static void main(String[] args) {
3.           int total = ( 2 - 1 ) * 2;
4.           total = total + ( 4 - 1 ) * 2;
5.           total = total + ( 3 - 1 ) * 2;
6.           System.out.print( total );
7.     }
8. }
```

(1) 12 - Correta (2) 10 (3) 14 (4) 11 (5) 13

Fonte: SEGALOTTO (2018)

Figura 32: Tarefa T4 - Processar Pontos v2.

```
1.  public class SoccerV2 {
2.     public static void main(String[] args) {
3.           Score points = new Score();
4.           int total = points.calculatePoints(3);
5.           total = total + points.calculatePoints(1);
6.           total = total + points.calculatePoints(2);
7.           System.out.print(total);
8.     }
9.  }
10. public class Score {
11.          public int calculatePoints ( int gols ) {
12.                return ( gols + 1 ) * 3;
13.          }
14.
15.}
```

(1) 25 (2) 24 (3) 29 (4) 21 (5) 27 - Correta

Fonte: SEGALOTTO (2018)

Figura 33: Tarefa T5 - Obter Calendário.

```
1. public class Dates {
2.     public static void main(String[] args) {
3.           int val = 90 + 7 + 365 - 30 - 1;
4.                 System.out.print( val );
5.           }
6.
7. }
```

(1) 12 - Correta (2) 10 (3) 14 (4) 11 (5) 13

Fonte: SEGALOTTO (2018)

Figura 34: Tarefa T6 - Obter Calendário v2.

```
1. public class DatesV2 {
2.     public static void main(String[] args) {
3.             int val = 100;
4.             val = new Week().remove(val);
5.             val = new Year().add(val);
6.             val = new Month().remove(val);
7.             val = new Day().add(val);
8.             System.out.print( val );
9.     }
10. }
11. public class Day {
12.    public int add ( int v ) {
13.            return v + 1;
14.    }
15. }
16. public class Month {
17.    public int remove ( int v ) {
18.            return v - 30;
19.    }
20. }
21. public class Week {
22.    public int remove ( int v ) {
23.            return v - 7;
24.    }
25. }
26. public class Year {
27.    public int add ( int v ) {
28.            return v + 365;
29.    }
30. }
```

(1) 452 (2) 458 (3) 428 (4) 459 (5) 429 - Correta

Fonte: SEGALOTTO (2018)

Figura 35: Tarefa T7 - Contador Elementos Fila.

```
1. public class Queue {
2.     public static void main(String[] args) {
3.         int[] tasks = { 3, 6, 4 };
4.         int total = 0;
5.         for ( int count = 0; count < tasks.length; count++ ) {
6.                 total = total + ( ( tasks[count] + 2 ) * 2 );
7.         }
8.         System.out.print( total );
9.     }
10. }
```

(1) 40 (2) 38 - Correta (3) 36 (4) 30 (5) 42

Fonte: SEGALOTTO (2018)

Figura 36: Tarefa T8 - Contador Elementos Fila v2.

```
1. public class QueueV2 {
2.     public static void main(String[] args) {
3.     int[] tasks = { 2, 7, 3 };
4.     int total = 0;
5.     Time clock = new Time();
6.        for ( int counter = 0; counter < tasks.length; counter++ ) 7.{
7.                  total = total + clock.delay ( tasks[counter] );
8.           }
9.     System.out.print( total );
10.    }
11. }
12. public class Time {
13.    public int delay ( int val ) {
14.          return ( val + 2 ) * 2;
15.    }
16. }
```

(1) 40 (2) 42 (3) 30 (4) 36 - Correta (5) 38

Fonte: SEGALOTTO (2018)

Figura 37: Tarefa T9 - Contador de Cartões.

```
1. public class Cards {
2.
3.     public static void main(String[] args) {
4.     int[] tasks = { 8, 6, 3, 9, 0, 1 };
5.     int amount = 0;
6.     for ( int counter = 0; counter < tasks.length; counter++ ){
7.                         switch ( tasks[counter] ) {
8.                         case 1:  amount = amount + 3;
9.                                break;
10.                        case 3:  amount = amount + 4;
11.                               break;
12.                        case 0:  amount = amount + 2;
13.                               break;
14.                        default: amount = amount + 1;
15.                               break;
16.                        }
17.          }
18.          System.out.print( amount );
19.          }
20.
21.
22.}
```

(1) 10 (2) 12 - Correta (3) 11 (4) 13 (5) 14

Fonte: SEGALOTTO (2018)

Figura 38: Tarefa T10 - Contador de Cartões v2.

```
1. public class CardsV2 {
2.     public static void main(String[] args) {
3.           // TODO Auto-generated method stub
4.           int[] tasks = { 4, 2, 7, 6, 3, 2 };
5.           int qtd = 0;
6.                 for ( int n = 0; n < tasks.length; n++ ) {
7.                       switch ( tasks[n] ) {
8.                       case 2:  TypeA a = new TypeA();
9.                             qtd = a.plus ( qtd );
10.                            break;
11.                      case 4:  TypeB b = new TypeB();
12.                            qtd = b.plus ( qtd );
13.                            break;
14.                      case 6:  TypeC c = new TypeC();
15.                            qtd = c.plus ( qtd );
16.                            break;
17.                      default: TypeD d = new TypeD();
18.                            qtd = d.plus ( qtd );
19.                            break;
20.                      }
21.                }
22.          System.out.print( qtd );
23.          }
24. }
25. public class TypeA {
26.    public int plus ( int vl ) {
27.          return vl + 2;
28.    }
29. }
30.
31. public class TypeB {
32.    public int plus ( int op ) {
33.          return op + 5;
34.    }
35. }
36. public class TypeC {
37.    public int plus ( int fy ) {
38.          return fy + 3;
39.    }
40. }
41. public class TypeD {
42.    public int plus ( int ts ) {
43.          return ts + 1;
44.    }
45. }
```

(1) 13	(2) 10	(3) 12	(4) 11	(5) 14 - Correta

Fonte: SEGALOTTO (2018)

APÊNDICE A – LISTA DE ARTIGOS SUBMETIDOS/PUBLICADOS EM RELAÇÃO A TESE

Capítulo	Referência	Situação
Capítulo 3 - Revisão da Literatura	GONÇALES, Lucian; FARIAS, Kleinner; DA SILVA, Bruno C; Jonathan Fessler. Measuring the cognitive load of software developers: a systematic mapping study.**In: Proceedings of the 27th International Conference on Program Comprehension.**. IEEE Press, 2019. p. 42-52.	**Publicado**
Capítulo 3 - Revisão da Literatura	GONCALES, Lucian. Towards the Measurement of Mental Effort in Software Engineering: A Research Agenda. **International Journal of Computer Applications.** Number 177(34):1-8, January 2020	**Publicado**
Capítulo 3 - Revisão da Literatura	GONÇALES, Lucian; FARIAS, Kleinner; DA SILVA, Bruno C; Jonathan Fessler. Measuring the cognitive load of software developers: An extended Systematic Mapping Study. **Information and Software Technology**. . March 2021	**Publicado**
Capítulo 4 e 5 - Abordagem/Avaliação	GONÇALES, Lucian; FARIAS, Kleinner; CogEff: a technique for measuring developers' Cognitive load based on comprehension tasks. **International Conf. on Software Engineering**. September 2021	**Em Avaliação**
Capítulo 6 - Machine Learning	GONÇALES, Lucian; FARIAS, Kleinner; KUPSSINSKÜ, Lucas; SEGALOTTO, Matheus. Evaluation of machine learning techniques to classify code comprehension based on developers' EEG data. **Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems (IHC 2020)**. Outubro 2020	**Publicado**
Capítulo 6 - Machine Learning	GONÇALES, Lucian; FARIAS, Kleinner; The effects of applying filters on EEG signals for classifying developers' code comprehension. **The Journal of Applied Research and Technology**. Agosto 2021	**Publicado**
Capítulo 6 - Machine Learning	GONÇALES, Lucian; FARIAS, Kleinner; KUPSSINSKÜ, Lucas; SEGALOTTO, Matheus. An empirical evaluation of machine learning techniques to classify code comprehension based on EEG data. **Expert Systems with Applications**. Junho 2021	**Publicado**

APÊNDICE B – LISTA DE ESTUDOS PRIMÁRIOS

S01 N. Peitek, J. Siegmund, S. Apel, C. Kästner, C. Parnin, A. Bethmann, T. Leich, G. Saake, & A. Brechmann "A Look into Programmers' Heads." IEEE Transactions on Software Engineering, 2018.

S02 M. Zimoch, R. Pryss, J. Schobel, & M. Reichert. "Eye Tracking Experiments on Process Model Comprehension: Lessons Learned." In Ent., Business-Process and Inf. Systems Modeling. BPMDS 2017. Lec. Notes in Business Inf. Proc., v 287.

S03 N. Nourbakhsh, Y. Wang, & F. Chen. "GSR and blink features for cognitive load classification," In IFIP Conf. on HCI, 2013, pp. 159–166.

S04 I. Crk, T. Kluthe, & A. Stefik. "Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes". ACM TOCHI. 23, 1, 2015.

S05 S. Fakhoury, Y. Ma, V. Arnaoudova, & O. Adesope "The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load." ICPC, 2018.

S06 T. Fritz & S. C. Müller, "Leveraging Biometric Data to Boost Software Developer Productivity," SANER, 2016, pp. 66–77.

S07 M. Borys, M. Plechawska-Wójcik, M. Wawrzyk & K. Wesołowska. "Classifying Cognitive Workload Using Eye Activity and EEG Features in Arithmetic Tasks." Information and Software Technology, 2017, pp. 90–105.

S08 S. C. Müller & T. Fritz, "Stuck and Frustrated or in Flow and Happy: Sensing Developers' Emotions and Progress," ICSE, 2015, pp. 688–699.

S09 I. Crk & T. Kluthe, "Assessing the contribution of the individual alpha frequency (IAF) in an EEG-based study of program comprehension," EMBC, 2016, pp. 4601–4604.

S10 D. Girardi, F. Lanubile, N. Novielli, & D. Fucci. 2018. "Sensing developers' emotions: the design of a replicated experiment," SEmotion, 2018, pp. 51–54.

S11 R. Petrusel, J. Mendling, & H. A. Reijers. "Task-specific visual cues for improving process model understanding." Information and Software Technology, v. 79, 2016, pp. 63–78.

S12 R. K. Minas, R. Kazman, & E. Tempero. "Neurophysiological Impact of Software design processes on software developers." In Int. Conf. on Augmented Cognition, 2017, pp. 56–64.

S13 R. Petrusel, J. Mendling & H. A. Reijers. "How visual cognition influences process model comprehension." Decision Support Systems, V. 96, 2017, pp. 1–16.

S14 M. Tallon, M. Winter, R. Pryss, K. Rakoczy, M. Reichert, M. W. Greenlee, & U. Frick. "Comprehension of business process models: Insight into cognitive strategies via eye tracking." Expert Systems with Applications, V. 136, 2019, pp. 145–158.

S15 N. Peitek, S. Apel, A. Brechmann, C. Parnin, & J. Siegmund. "CodersMUSE: multi-modal data exploration of program-comprehension experiments." ICPC, 2019, pp. 126–129.

S16 T. Baum, K. Schneider, & A. Bacchelli. "Associating working memory capacity and code change ordering with code review performance." Empirical Software Engineering, 2019, v. 24, n. 4, pp. 1762–1798.

S17 S. Lee, A., Matteson, D. Hooshyar, S. Kim, J. Jung, G. Nam, & H. Lim. "Comparing programming language comprehension between novice and expert programmers using EEG analysis." BIBE, 2016, pp. 350–355.

S18 K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, & T. Fritz. "Tracing software developers' eyes and interactions for change tasks." ESEC/FSE, 2015, pp. 202–213.

S19 M. Züger & T. Fritz. "Interruptibility of Software Developers and its Prediction Using Psycho-Physiological Sensors." CHI. 2015, pp. 2981–2990.

S20 N. Peitek, J. Siegmund, C. Parnin, S. Apel, & A. Brechmann. "Toward conjoint analysis of simultaneous eye-tracking and fMRI data for program-comprehension studies." EMIP, 2018.

S21 M. Behroozi & C. Parnin. "Can we predict stressful technical interview settings through eye-tracking?." EMIP, 2018.

S22 J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, & A. Brechmann, "Measuring neural efficiency of program comprehension." In ESEC/FSE, 2017, pp. 140–150.

S23 J. S. Moll'eri, I. Nurdiani, F. Fotrousi, & K. Petersen. "Experiences of studying Attention through EEG in the Context of Review Tasks." EASE, 2019, pp. 313–318.

S24 A. Duraisingam, R. Palaniappan, & S. Andrews "Cognitive task difficulty analysis using EEG and data mining." ICEDSS, 2017, pp. 52–57.

S25 D. Fucci, D. Girardi, N. Novielli, L. Quaranta, and F. Lanubile. "A replication study on code comprehension and expertise using lightweight biometric sensors." In ICPC, 2019.

S26 J. Castelhano, I. C. Duarte, C. Ferreira, J. Duraes, H. Madeira, & M. Castelo-Branco. "The role of the insula in intuitive expert bug detection in computer code: an fMRI study." Brain Imaging and Behavior, V. 13, I. 3, pp. 623–637.

S27 K. Kevic, B.M. Walters, T.R. Shaffer, B. Sharif, D.C. Shepherd, & T. Fritz "Eye gaze and interaction contexts for change tasks – Observations and potential." JSS, V. 128, 2017, pp. 252–266.

S28 M. Konopka. "Combining eye tracking with navigation paths for identification of cross-language code dependencies." ESEC/FSE, 2015, pp. 1057–1059

S29 S. C. Müller & T. Fritz. "Using (bio) metrics to predict code quality online." ICSE. 2016, pp. 452–463.

S30 S. C. Müller "Measuring software developers' perceived difficulty with biometric sensors." ICPC, 2015, pp. 887–890.

S31 N. Peitek, J. Siegmund, C. Parnin, S. Apel, & A. Brechmann "Beyond gaze: preliminary analysis of pupil dilation and blink rates in an fMRI study of program comprehension." In EMIP, 2018.

S32 M. Züger & T. Fritz "Sensing and supporting software developers' focus." In ICPC, 2018.

S33 S. Lee, D. Hooshyar, H. Ji, K. Nam, & H. Lim. "Mining biometric data to predict programmer expertise and task difficulty.", Cluster Computing, 2017, pp. 1–11.

S34 R. Couceiro, G. Duarte, J. Durães, J. Castelhano, C. Duarte, C. Teixeira, M. C. Branco, P. Carvalho, & H. Madeira "Biofeedback augmented software engineering: monitoring of programmers' mental effort." ICSE: NIER, 2019, pp. 37–40.

S35 T. Ishida, & H. Uwano "Synchronized analysis of eye movement and EEG during program comprehension." EMIP, 2019, pp. 26–32.

S36 M. P. Uysal. "Towards the use of a novel method: The first experiences on measuring the cognitive load of learned programming skills." Turkish Online Journal of Distance Education, v.14, n. 1, 2013, pp. 166–184.

S37 T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, & M. Züger "Using psycho-physiological measures to assess task difficulty in software development." ICSE, 2014, pp. 402–413.

S38 F. Schaule, J. O. Johanssen, B. Bruegge, & V. Loftness "Employing Consumer Wearables to Detect Office Workers' Cognitive Load for Interruption Management." Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Tech., v. 2, n. 1, 2018, pp. 32:1–32:20.

S39 M. Züger, S. C. Müller, A. N. Meyer, & T. Fritz "Sensing interruptibility in the office: A field study on the use of biometric and computer interaction sensors." CHI, 2018, pp. 591.

S40 Y. Huang, X. Liu, R. Krueger, T. Santander, X. Hu, K. Leach, & W. Weimer "Distilling neural representations of data structure manipulation using fMRI and fNIRS." ICSE, 2019, pp. 396–407.

S41 M. V. Kosti, K. Georgiadis, D. A. Adamos, N. Laskaris, D. Spinellis, & L. Angelis "Towards an affordable brain computer interface for the assessment of programmers' mental workload." Int. Journal of HCI, 2018, v. 115, pp. 52–66.

S42 M. Behroozi, A. Lui, I. Moore, D. Ford, & C. Parnin. "Dazed: measuring the cognitive load of solving technical interview problems at the whiteboard." ICSE: NIER, 2018, pp. 93–96.

S43 J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, & A. Brechmann "Understanding understanding source code with functional magnetic resonance imaging." In ICSE, 2014, pp. 378–389.

S44 M. K. C. Yeh, D. Gopstein, Y. Yan, & Y. Zhuang “Detecting and comparing brain activity in short program comprehension using EEG.” IEEE Front. in Education Conf., 2017, pp. 1–50.

S45 B. Floyd, T. Santander, & W. Weimer “Decoding the representation of code in the brain: an fMRI study of code review and expertise.” ICSE, 2017, pp. 175–186.

S46 N. Peitek, J. Siegmund, C. Parnin, S. Apel, J. C. Hofmeister, A. Brechmann “Simultaneous measurement of program comprehension with fMRI and eye tracking: a case study.” ESEM, 2018.

S47 B. Sharif, & J. I. Maletic, “An eye tracking study on camelcase and under_score identifier styles.” ICPC, 2010, pp. 196–205.

S48 Y. Ikutani & H. Uwano “Brain activity measurement during program comprehension with NIRS” SNPD, 2014, pp. 1–6.

S49 J. Duraes, H. Madeira, J. Castelhano, C. Duarte, & M. C. Branco “WAP: Understanding the Brain at Software Debugging” ISSRE, 2016, pp. 87–92.

S50 T. Nakagawa, Y. Kamei, H. Uwano, A. Monden, K. Matsumoto, D. M. & German “Quantifying programmers’ mental workload during program comprehension based on cerebral blood flow measurement: a controlled experiment.” ICSE, 2014, pp. 448-451.

S51 S. Radevski, H. Hata & K. Matsumoto “Real-time monitoring of neural state in assessing and improving software developers’ productivity” CHASE, 2015, pp. 93–96.

S52 A. Yamamoto, H. Uwano, & Y. Ikutani. “Programmer’s electroencephalogram who found implementation strategy.” ACIT-CSII-BCD, 2016, pp. 164-168.

S53 C. Parnin “Subvocalization-toward hearing the inner thoughts of developers” ICPC, 2011, pp. 197–200.

S54 J. Siegmund, A. Brechmann, S. Apel, C. Kästner, J. Liebig, T. Leich, & G. Saake. “Toward measuring program comprehension with functional magnetic resonance imaging.” ESEC/FSE, 2012, pp. 1-4.

S55 R. Ikramov, V. Ivanov, S. Masyagin, R. Shakirov, I. Sirazidtinov, G. Succi, & O. Zufarova. “Initial evaluation of the brain activity under different software development situations.” SEKE, 2018.

S56 S. Fakhoury, D. Roy, Y. Ma, V. Arnaoudova, & O. Adesope. “Measuring the impact of lexical and structural inconsistencies on developers’ cognitive load during bug localization” Emp. Soft. Engineering, 1-39, 2019.

S57 R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer, & K. Leach “Neurological Divide: An fMRI Study of Prose and Code Writing”, ICSE, 2020.

S58 R. Palaniappan, A. Duraisingam, N. Chinnaiah, & M. Murugappan “Predicting Java Computer Programming Task Difficulty Levels Using EEG for Educational Environments.” HCI, 2019, pp. 446–460.

S59 R. Couceiro, G. Duarte, J. Durães, J. Castelhano, C. Duarte, C. Teixeira, M. C. Branco, P. Carvalho & H. Madeira “Pupillography as Indicator of Programmers’ Mental Effort and Cognitive Overload” DSN, 2019, pp. 638–644.

S60 J. Medeiros, R. Couceiro, J. Castelhano, M. Castelo Branco, G. Duarte, C. Duarte, J. Durães, H. Madeira, P. Carvalho, C. Teixeira, “Software code complexity assessment using EEG features.” International Conference of the IEEE EMBC, 2019, pp. 1413–1416.

S61 R. Couceiro, R. Barbosa, J. Duráes, G. Duarte, J. Castelhano, C. Duarte, C. Teixeira, N. Laranjeiro, J. Medeiros, P. Carvalho, H. Madeira. “Spotting Problematic Code Lines using Nonintrusive Programmers’ Biofeedback” ISSRE, pp. 93–103, 2019.

S62 T. Ishida & H. Uwano “Time series analysis of programmer’s EEG for debug state classification” Programming, 2019, pp. 1–7.

S63 D. Roy, S. Fakhoury & V. Arnaoudova “VITALSE: Visualizing Eye Tracking and Biometric Data.” ICSE, 2020.

Extensões dos estudos primários: QP1 de [S30] estendeu [S37], [S08] estende QP2 de [S30], [S29] estende QP3 do estudo [S30], [S06] estende desafios e trabalhos futuros de [S30], [S01] estende [S22], [S22] replicou [S43], [S39] estende [S19], [S56] estende [S05], [S27] estende [S18].

APÊNDICE C – PRECISÃO, RECALL E F-MEASURE POR CLASSE

Tabela 42: Classificador *K-Nearest Neighbor*: resultados da precisão, *recall* e *f-measure*.

	Correta			Incorreta			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,95	0,84444444	0,89411765	0,8490566	0,97826087	0,90909091	0,97435897	0,92682927	0,95
Fold 2	0,875	0,74468085	0,8045977	0,84	0,89361702	0,86597938	0,80952381	0,89473684	0,85
Fold 3	0,88636364	0,79591837	0,83870968	0,77777778	0,89361702	0,83168317	0,82352941	0,77777778	0,8
Fold 4	0,92857143	0,78	0,84782609	0,67307692	0,94594595	0,78651685	0,97368421	0,82222222	0,89156627
Fold 5	0,75	0,75	0,75	0,82	0,85416667	0,83673469	0,89130435	0,85416667	0,87234043
Fold 6	0,97435897	0,86363636	0,91566265	0,87272727	1	0,93203883	0,97368421	0,925	0,94871795
Fold 7	0,825	0,73333333	0,77647059	0,77083333	0,925	0,84090909	0,84090909	0,78723404	0,81318681
Fold 8	0,875	0,77777778	0,82352941	0,73809524	0,91176471	0,81578947	0,98275862	0,91935484	0,95
Fold 9	0,92307692	0,7826087	0,84705882	0,82142857	0,95833333	0,88461538	0,89189189	0,86842105	0,88
Fold 10	0,91666667	0,78571429	0,84615385	0,81132075	0,95555556	0,87755102	0,95348837	0,91111111	0,93181818
Todas	0,890403763	0,785811412	0,834412644	0,797431646	0,931626113	0,858090879	0,911513293	0,868685382	0,888762964

Fonte: Elaborado pelo autor.

Tabela 43: Classificador *Neural Network*: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,82978723	0,86666667	0,84782609	0,82978723	0,84782609	0,83870968	0,97368421	0,90243902	0,93670886
Fold 2	0,80851064	0,80851064	0,80851064	0,84444444	0,80851064	0,82608696	0,85	0,89473684	0,87179487
Fold 3	0,76923077	0,81632653	0,79207921	0,81395349	0,74468085	0,77777778	0,86486486	0,88888889	0,87671233
Fold 4	0,82978723	0,78	0,80412371	0,73333333	0,89189189	0,80487805	0,975	0,86666667	0,91764706
Fold 5	0,88571429	0,86111111	0,87323944	0,88235294	0,9375	0,90909091	0,97826087	0,9375	0,95744681
Fold 6	0,75471698	0,90909091	0,82474227	0,86363636	0,79166667	0,82608696	0,97142857	0,85	0,90666667
Fold 7	0,81081081	0,66666667	0,73170732	0,7	0,875	0,77777778	0,88888889	0,85106383	0,86956522
Fold 8	0,96428571	0,75	0,84375	0,75555556	1	0,86075949	0,98305085	0,93548387	0,95867769
Fold 9	0,86363636	0,82608696	0,84444444	0,82978723	0,8125	0,82105263	0,85365854	0,92105263	0,88607595
Fold 10	0,8372093	0,85714286	0,84705882	0,8	0,88888889	0,84210526	0,97435897	0,84444444	0,9047619
Todas	0,835368932	0,814160235	0,821748194	0,805285058	0,859846503	0,82843255	0,931319576	0,889227619	0,908605736

Fonte: Elaborado pelo autor.

Tabela 44: Classificador *Naïve Bayes*: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem Resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,85714286	0,13333333	0,23076923	0,47126437	0,89130435	0,61654135	0,55263158	0,51219512	0,53164557
Fold 2	1	0,12765957	0,22641509	0,52380952	0,93617021	0,67175573	0,4047619	0,44736842	0,425
Fold 3	1	0,16326531	0,28070175	0,51086957	1	0,67625899	0,46875	0,41666667	0,44117647
Fold 4	1	0,16	0,27586207	0,47945205	0,94594595	0,63636364	0,60784314	0,68888889	0,64583333
Fold 5	1	0,02777778	0,05405405	0,46464646	0,95833333	0,62585034	0,53125	0,35416667	0,425
Fold 6	1	0,18181818	0,30769231	0,51898734	0,85416667	0,64566929	0,37777778	0,425	0,4
Fold 7	1	0,08888889	0,16326531	0,41758242	0,95	0,58015267	0,62162162	0,4893617	0,54761905
Fold 8	1	0,08333333	0,15384615	0,3875	0,91176471	0,54385965	0,65306122	0,51612903	0,57657658
Fold 9	1	0,13043478	0,23076923	0,54761905	0,95833333	0,6969697	0,5	0,55263158	0,525
Fold 10	0,8	0,0952381	0,17021277	0,46666667	0,93333333	0,62222222	0,64864865	0,53333333	0,58536585
Todas	0,965714286	0,119174927	0,209358796	0,478839745	0,933935188	0,631564358	0,536634589	0,493574141	0,510321685

Fonte: Elaborado pelo autor.

Tabela 45: Classificador *Random Forest*: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem Resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,76363636	0,93333333	0,84	0,87179487	0,73913043	0,8	0,94736842	0,87804878	0,91139241
Fold 2	0,80851064	0,80851064	0,80851064	0,84090909	0,78723404	0,81318681	0,82926829	0,89473684	0,86075949
Fold 3	0,86956522	0,81632653	0,84210526	0,76470588	0,82978723	0,79591837	0,88571429	0,86111111	0,87323944
Fold 4	0,95238095	0,8	0,86956522	0,6875	0,89189189	0,77647059	0,95238095	0,88888889	0,91954023
Fold 5	0,84848485	0,77777778	0,8115942	0,82352941	0,875	0,84848485	0,9375	0,9375	0,9375
Fold 6	0,87234043	0,93181818	0,9010989	0,87234043	0,85416667	0,86315789	0,92105263	0,875	0,8974359
Fold 7	0,8372093	0,8	0,81818182	0,73913043	0,85	0,79069767	0,88372093	0,80851064	0,84444444
Fold 8	0,88235294	0,83333333	0,85714286	0,80555556	0,85294118	0,82857143	0,9516129	0,9516129	0,9516129
Fold 9	0,875	0,76086957	0,81395349	0,8	0,83333333	0,81632653	0,85714286	0,94736842	0,9
Fold 10	0,85	0,80952381	0,82926829	0,80851064	0,84444444	0,82608696	0,84444444	0,84444444	0,84444444
Todos	0,855948069	0,827149317	0,839142068	0,801397631	0,835792921	0,81589011	0,901020571	0,888722202	0,894036925

Fonte: Elaborado pelo autor.

Tabela 46: Classificador *Support Vector Machine*: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,6875	0,73333333	0,70967742	0,75555556	0,73913043	0,74725275	0,79487179	0,75609756	0,775
Fold 2	0,7173913	0,70212766	0,70967742	0,76744186	0,70212766	0,73333333	0,69767442	0,78947368	0,74074074
Fold 3	0,75	0,67346939	0,70967742	0,73469388	0,76595745	0,75	0,69230769	0,75	0,72
Fold 4	0,78723404	0,74	0,7628866	0,62790698	0,72972973	0,675	0,80952381	0,75555556	0,7816092
Fold 5	0,7	0,58333333	0,63636364	0,67272727	0,77083333	0,7184466	0,82978723	0,8125	0,82105263
Fold 6	0,70833333	0,77272727	0,73913043	0,8372093	0,75	0,79120879	0,85365854	0,875	0,86419753
Fold 7	0,6	0,6	0,6	0,65384615	0,85	0,73913043	0,74285714	0,55319149	0,63414634
Fold 8	0,8125	0,72222222	0,76470588	0,66666667	0,88235294	0,75949367	0,90909091	0,80645161	0,85470085
Fold 9	0,71153846	0,80434783	0,75510204	0,78571429	0,6875	0,73333333	0,81578947	0,81578947	0,81578947
Fold 10	0,75757576	0,5952381	0,66666667	0,69090909	0,84444444	0,76	0,79545455	0,77777778	0,78651685
Todas	0,723207289	0,692679913	0,705388752	0,719267105	0,772207598	0,74071989	0,794101555	0,769183715	0,779375361

Fonte: Elaborado pelo autor.

Tabela 47: Classificador aleatório/*Naive*: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,29545455	0,28888889	0,29213483	0,2962963	0,34782609	0,32	0,32352941	0,26829268	0,29333333
Fold 2	0,44186047	0,40425532	0,42222222	0,34090909	0,31914894	0,32967033	0,42222222	0,5	0,45783133
Fold 3	0,31578947	0,36734694	0,33962264	0,30555556	0,23404255	0,26506024	0,12820513	0,13888889	0,13333333
Fold 4	0,35714286	0,3	0,32608696	0,325	0,35135135	0,33766234	0,38	0,42222222	0,4
Fold 5	0,28301887	0,41666667	0,33707865	0,425	0,35416667	0,38636364	0,28205128	0,22916667	0,25287356
Fold 6	0,35135135	0,29545455	0,32098765	0,33962264	0,375	0,35643564	0,30952381	0,325	0,31707317
Fold 7	0,4	0,35555556	0,37647059	0,24444444	0,275	0,25882353	0,36170213	0,36170213	0,36170213
Fold 8	0,22916667	0,30555556	0,26190476	0,35714286	0,44117647	0,39473684	0,5	0,33870968	0,40384615
Fold 9	0,32432432	0,26086957	0,28915663	0,2	0,16666667	0,18181818	0,2	0,28947368	0,23655914
Fold 10	0,2745098	0,33333333	0,30107527	0,34146341	0,31111111	0,3255814	0,35	0,31111111	0,32941176
Todas	0,327261836	0,332792639	0,32667402	0,31754343	0,317548985	0,315615214	0,325723398	0,318456706	0,31859639

Fonte: Elaborado pelo autor.

Tabela 48: Classificador *K-Nearest Neighbor* treinado com dados psicofisiológicos e os resultados da CogEff: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,97	0,8888888	0,927674787	0,892235456	0,9999999	0,943049039	0,9888888	0,93152156	0,959348332
Fold 2	0,986	0,8111111	0,890045746	0,8666666	0,9302451	0,89733108	0,8665654	0,962121	0,911846634
Fold 3	0,89333	0,899999999	0,896652596	0,77777778	0,945321	0,853403969	0,86215645	0,8715212	0,866813532
Fold 4	0,945554	0,8333333	0,885903941	0,73333333	0,988888	0,842150213	0,985612	0,96121221	0,973259202
Fold 5	0,778888	0,866666667	0,820436149	0,877777777	0,911111	0,894133832	0,9615122	0,9412655	0,951281131
Fold 6	0,987777	0,922222222	0,953874629	0,93333333	0,9666	0,949675425	0,99999	0,9756121	0,987650645
Fold 7	0,866666	0,822222222	0,843859333	0,799999991	0,988888	0,884471689	0,892121	0,892123	0,892122
Fold 8	0,9222222	0,844233415	0,88150621	0,7777777	0,945231	0,853367244	1	0,96	0,979591837
Fold 9	0,973335555	0,84423221	0,904198724	0,8666666	0,975121	0,91770061	0,96152151	0,9712156	0,966344244
Fold 10	0,96666661	0,8	0,875471675	0,866666	0,999999	0,928570615	0,96120012	0,9321231	0,946438332
Todas	0,929043937	0,853290994	0,887962379	0,839223456	0,9651404	0,896385371	0,947956748	0,939871527	0,943469589

Fonte: Elaborado pelo autor.

Tabela 49: Classificador *Neural Network* treinado com dados psicofisiológicos e da CogEff: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,877777	0,95	0,912461586	0,932121	0,875115	0,902718924	0,988888	0,9856551	0,987268903
Fold 2	0,8456231	0,87	0,857638367	0,8741512	0,861521	0,867790146	0,892121	0,9512132	0,920719934
Fold 3	0,822222	0,9315111	0,873461212	0,91512151	0,821211	0,865626654	0,892112	0,9	0,89603864
Fold 4	0,87213121	0,923121	0,896901984	0,7877454	0,9321231	0,853874217	0,999999	0,89564115	0,944947546
Fold 5	0,933333	0,892151	0,912277477	0,9252115	0,95121	0,938030641	1	0,96	0,979591837
Fold 6	0,75471698	0,99999	0,860211353	0,9315511	0,8612111	0,895001186	0,982131	0,89544	0,936784156
Fold 7	0,87121	0,82121	0,845471413	0,86	0,86121	0,860604575	0,966211	0,923212	0,94422222
Fold 8	1	0,83333	0,909088926	0,86121231	1	0,925431565	0,9112021	0,984511	0,94643909
Fold 9	0,93321215	0,905112	0,918947309	0,84521231	0,93212	0,886541356	0,958451	0,933333	0,945725249
Fold 10	0,9412151	0,932222	0,936696965	0,86	0,96551	0,90970589	1	0,8745112	0,933055188
Todas	0,8851441	0,90586471	0,89231566	0,879233	0,90612	0,89053252	0,959112	0,9303517	0,94347928

Fonte: Elaborado pelo autor.

Tabela 50: Classificador *Random Forest* treinado com dados psicofisiológicos e da CogEff: resultados da precisão, *recall* e *f-measure*.

	Corretas			Incorretas			Sem Resposta		
	Precisão	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**	**Precisão**	**Recall**	**F-measure**
Fold 1	0,7865312	0,9651512	0,866734211	0,922222	0,76126165	0,834046997	0,97851521	0,8888888	0,900523766
Fold 2	0,83221516	0,833333	0,832773705	0,9321213	0,799999	0,861021152	0,84	0,9	0,850380687
Fold 3	0,92156156	0,8232155	0,869616844	0,911111	0,866666	0,88833293	0,8899999	0,8773295	0,889165634
Fold 4	0,966666	0,86666	0,913935389	0,7212313	0,9212131	0,809047444	0,9777777	0,89	0,885445956
Fold 5	0,87123121	0,799551	0,833853486	0,8621231	0,9	0,880654467	0,9666666	0,9455465	0,921658151
Fold 6	0,888888	0,988888	0,936225276	0,922222	0,9123138	0,917241143	0,9321513	0,9	0,924636117
Fold 7	0,86154564	0,833333	0,847204509	0,789512	0,89	0,836749818	0,9	0,8111111	0,867223164
Fold 8	0,8999999	0,9121512	0,90603481	0,8231212	0,8989116	0,85934855	0,9888888	0,9651561	0,919579032
Fold 9	0,9111111	0,7895641	0,845994127	0,8562133	0,888888	0,872244755	0,8966666	0,9615615	0,884287092
Fold 10	0,899999	0,8621021	0,880643032	0,83231156	0,966666	0,894471731	0,9111111	0,86521512	0,902714746
Todas	0,88397488	0,8673949	0,873302	0,8572189	0,880592	0,86531590	0,9281777	0,9004809	0,89456143

Fonte: Elaborado pelo autor.