



Prediction of Cognitive Strain During the Development Phase of the Software Development Life Cycle Using Machine Learning Models

Surya Bhagavan Raju Pericherla
Mounish Porandla

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:

Author(s):

Surya Bhagavan Raju Pericherla

E-mail: supe22@student.bth.se

Mounish Porandla

E-mail: mopo21@student.bth.se

Abstract

Background: Cognitive strain among software developers during the coding phase of the Software Development Life Cycle (SDLC) can adversely affect productivity and well-being. Understanding the factors contributing to cognitive strain and developing effective strategies to manage it are crucial for optimizing performance in software development.

Objectives: This study aims to identify the key factors influencing cognitive strain during the coding phase of the SDLC, quantify these factors for effective analysis, It also seeks to develop and evaluate machine learning models to predict cognitive strain, thereby exploring how identification and management can improve productivity and developer well-being.

Methods: A systematic approach was employed, beginning with a theoretical foundation established through a literature review, followed by empirical data collection via a structured questionnaire. Insights from experienced developers refined the identified factors, which were then used to develop predictive machine learning models, including Random Forest, LSTM neural networks, Logistic Regression, and K-Nearest Neighbors. These models assessed cognitive strain and informed the formulation of evidence-based management strategies, although their practical implementation was beyond the study's scope. By quantifying cognitive strain and leveraging predictive analytics, this research provides a structured methodology for identifying, analyzing, and mitigating cognitive strain, ultimately contributing to a more sustainable and productive software development process.

Results: Significant correlations were found between cognitive strain and factors like high task complexity, extended work hours, poor sleep quality, frequent multitasking, and high deadline pressure. The Random Forest model achieved the highest performance with accuracy close to 0.991, indicating excellent predictive capabilities. LSTM performed moderately well with an accuracy of 0.808, while LR and KNN had lower accuracies around 0.62. Based on these findings, strategies such as workload balancing, expertise-based task allocation, flexible scheduling, reducing multitasking, and providing stress management resources were proposed.

Conclusions: Identifying cognitive strain through predictive modeling enables organizations to implement targeted interventions that enhance productivity and developer well-being during the coding phase of the SDLC. The Random Forest model proved particularly effective in predicting cognitive strain. The proposed strategies, supported by empirical data and existing literature, offer actionable insights for proactively addressing cognitive strain. Future research should focus on testing these interventions in practical settings, expanding the dataset, and exploring additional factors influencing cognitive strain.

Keywords: Cognitive Strain, Software Development Life Cycle, Machine Learning, Productivity, Developer Well-being.

Acknowledgments

We would like to express our heartfelt gratitude to everyone who contributed to the successful completion of this thesis.

Firstly, we extend our sincere appreciation to the Department of Software Engineering at Blekinge Institute of Technology (BTH) for providing a stimulating academic environment and invaluable resources. The support and guidance from the faculty and the department have been crucial to the progress and completion of this research.

We are deeply thankful to our parents and siblings for their unwavering encouragement and support throughout this journey. Their belief in us has been a constant source of strength, motivating me to push forward, even in challenging moments.

We also wish to acknowledge our friends, family, and all those who participated in interviews, surveys and discussions for their patience, understanding, and thoughtful contributions. Their input and encouragement played a significant role in shaping the outcomes of this thesis, and we are sincerely grateful for their support.

Contents

Abstract	i
Acknowledgments	iii
1 Introduction	1
1.1 Research Scope and Focus	2
1.2 Aim and Research Objectives	3
1.3 Significance of the Study	4
1.4 Implications and Applications	4
1.5 Ethical Considerations	5
1.6 Outline	5
2 Background	7
2.1 Software Development Life Cycle (SDLC)	7
2.2 Types of SDLC Methodologies	8
2.3 The Significance of Cognitive Strain in SDLC	9
2.4 Predictive Modeling for Cognitive Strain Using Machine Learning Models	10
2.4.1 Random Forest (RF)	10
2.4.2 Long Short-Term Memory (LSTM)	11
2.4.3 Logistic Regression (LR)	11
2.4.4 K-Nearest Neighbors (KNN)	12
2.4.5 Rationale for Model Selection	12
2.4.6 Exclusion of Other Models	13
2.5 Challenges of Cognitive Strain in the Development/Coding Phase of SDLC	13
2.6 Relevance to the Current Research	14
3 Related Work	15
3.1 Research Gap	17
4 Method	19
4.1 Methodology for RQ1	19
4.1.1 Preliminary Research	21
4.1.2 Initial Parameters and Questionnaire Design	22
4.1.3 Consultation with Experienced Developers	23
4.1.4 Feedback and Justification for Parameter Refinement	24
4.1.5 Justification for Removing Certain Parameters	24

4.1.6	Finalized Parameters	25
4.1.7	Justification for Quantifying Subjective Measures	27
4.1.8	Data Collection	28
4.1.9	Threats to Validity for RQ1:	28
4.2	Methodology for RQ2	29
4.2.1	Data Preparation and Augmentation	29
4.2.2	Apply Machine Learning Models	31
4.2.3	Performance Metrics	32
4.2.4	Integration of Model Insights	33
4.2.5	Literature Review for Strategy Development	33
4.2.6	Threats to Validity for RQ2:	34
5	Results and Analysis	36
5.1	Results of RQ1	36
5.1.1	Results of preliminary research for Factors, Theories, and Models	36
5.1.2	Finalized Key Factors Influencing Cognitive Strain	38
5.1.3	Cognitive Strain Distribution Across Factors	38
5.1.4	Patterns and Insights	39
5.2	Results for RQ2	40
5.2.1	Machine Learning Model Performance	40
5.2.2	Performance Comparison	40
5.2.3	Overall Insights	41
5.3	Confusion Matrix Analysis	41
5.3.1	Random Forest	41
5.3.2	Logistic Regression	42
5.3.3	LSTM	42
5.3.4	K-Nearest Neighbors (KNN)	43
5.3.5	Summary of Confusion Matrix Analyses	44
5.3.6	Feature Importance Analysis	44
5.3.7	Strategies for Managing Cognitive Load in Software Development	45
6	Discussion	46
6.1	Interpretation of Key Findings	46
7	Conclusion and Future Work	50
7.1	Conclusion	50
7.2	Future Work	52
7.3	Final Remarks	53
	References	54
	A Supplemental Information	68

List of Figures

4.1	Research flow diagram for RQ1	20
4.2	Research flow diagram for RQ2	30
5.1	Comparative Analysis of Machine Learning Models	41
5.2	Confusion Matrix of Random Forest Model	41
5.3	Confusion Matrix of Logistic Regression Model	42
5.4	Confusion Matrix of LSTM Model	43
5.5	Confusion Matrix of KNN Model	43

List of Tables

4.1	Questionnaire Parameters and Their Purpose	26
5.1	Factors identified from Literature Review	37
5.2	Cognitive Strain Distribution Across Factors	38
5.3	Cognitive Strain Distribution Across Numeric Variables	39
5.4	Performance Metrics of Different Models	40
5.6	Strategies identified from Literature Review	45

The Software Development Life Cycle (SDLC) is a foundational framework that outlines the processes involved in creating software systems, including planning, designing, developing, testing, and deploying. Various methodologies like Agile, Waterfall, and DevOps have been developed to provide structured strategies for managing and executing software projects effectively. These methodologies aim to enhance efficiency, ensure quality, and meet the evolving needs of users and stakeholders.

As software systems grow in complexity and the demand for rapid delivery increases, developers face significant challenges during the development phase. One critical challenge is the mental effort required to perform complex tasks—commonly referred to as cognitive strain. Cognitive strain encompasses the mental load experienced when individuals are required to process complex information, solve intricate problems, or manage multiple tasks simultaneously. In software development, this can manifest when developers work with complicated codebases, debug challenging issues, or adapt to changing project requirements.

In Agile methodologies, for example, the iterative nature and continuous adaptation to evolving requirements demand constant focus and effective communication within teams. Developers must manage multiple tasks simultaneously, often under tight deadlines, which can be overwhelming, especially in high-pressure environments where rapid delivery is expected [1]. Research indicates that cognitive load is a significant factor in software development, particularly in large-scale projects where task complexity can exacerbate mental strain [2] [3]. The cognitive demands during the Agile development process can lead to decreased efficiency and increased error rates, as the mental effort required to handle multiple tasks and shifting priorities is substantial. Furthermore, the lack of adequate support systems to manage this cognitive load can result in burnout, affecting individual developers and jeopardizing project outcomes [4].

In modern software development, addressing both technical and human factors is essential for optimizing productivity. Cognitive strain, a key human factor, refers to the mental effort required to solve complex problems, maintain focus, and manage multiple tasks or streams of information. Developers often experience cognitive strain when working with intricate codebases, facing tight deadlines, debugging challenging issues, or handling frequent interruptions. Research indicates that such interruptions can exacerbate cognitive load, leading to increased errors and decreased productivity [5]. These pressures can significantly hinder a developer's decision-making ability, clarity of thought, and the quality of their output.

As software systems become more complex, the cognitive demands placed on

developers have increased correspondingly. Understanding and predicting cognitive strain during the development phase has emerged as a critical area of research. Traditional approaches to predicting cognitive strain have primarily focused on task complexity and workload estimation. Techniques like Function Point Analysis (FPA) and code complexity metrics (e.g., Cyclomatic Complexity, Halstead Metrics) estimate the mental effort required for specific tasks. Project management tools like Gantt charts and Agile methodologies utilizing story points help teams forecast periods of excessive workload that may surpass developer capacity, signaling potential cognitive strain [6].

Self-assessment tools such as the NASA Task Load Index (NASA-TLX) and Likert-scale surveys provide insights into developers' experiences of stress, mental fatigue, and frustration [7]. However, reliance on self-reported measures introduces variability, as individual perceptions of cognitive load can differ significantly, potentially obscuring real-time assessments of cognitive strain. Moreover, traditional metrics and self-reported assessments often fail to capture the dynamic and fluctuating nature of cognitive load during the coding phase.

To address these challenges, it is crucial to move beyond static metrics and self-reported assessments of cognitive strain. Instead, more dynamic, real-time methods for monitoring and predicting cognitive load during the development phase are needed. By leveraging advanced techniques such as machine learning models and real-time data analytics, we can better understand, thereby aiding to mitigate the factors contributing to developer strain. These approaches can offer more accurate, continuous feedback, allowing teams to intervene before cognitive overload leads to errors, burnout, or diminished productivity. This research will focus on identifying key indicators of cognitive strain, quantify these factors for effective analysis during the development phase and exploring effective strategies for reducing its impact, thereby improving both developer well-being and overall project success. The following sections will outline the research scope, focus, and the specific aims and questions guiding this study.

1.1 Research Scope and Focus

This research focuses on identifying key factors influencing cognitive strain during the coding phase of the SDLC and exploring effective strategies to manage it. Through comprehensive data collection from software developers, we aim to quantify factors such as task complexity, work hours, task duration, stress levels, sleep quality, multitasking, and deadline pressure. By analyzing these factors, we seek to understand their impact on cognitive strain and developer performance.

Building upon this understanding, we develop predictive models to identify cognitive strain among developers. While machine learning models like Random Forest, Long Short-Term Memory (LSTM), Logistic Regression, and K-Nearest Neighbors are employed, the emphasis is on how these models can be utilized to gain actionable insights rather than on the technical aspects alone. The models serve as tools to enhance our comprehension of cognitive strain and to facilitate the development of targeted strategies and interventions.

The research also focuses on proposing evidence-based strategies to mitigate cog-

nitive strain. By integrating findings from our data analysis with existing literature, we aim to offer practical recommendations that organizations can implement to improve productivity and developer well-being during the coding phase.

1.2 Aim and Research Objectives

Aim: The aim of this research is to identify and quantify the key factors influencing cognitive strain during the coding phase of the SDLC, develop effective predictive models to identify cognitive strain among developers, and propose strategies and interventions to improve productivity and developer well-being.

Objectives: To achieve the aim, the following objectives are established.

- Collect data from software developers to identify and quantify key factors influencing cognitive strain during the coding phase of the SDLC.
- Analyze the collected data to determine the relationships between the identified factors and cognitive strain, highlighting the most impactful contributors.
- Develop and evaluate predictive models to identify cognitive strain among developers, utilizing machine learning techniques as tools for analysis rather than the primary focus.
- Propose evidence-based strategies and interventions to manage cognitive strain, improve productivity, and enhance developer well-being, based on the findings from data analysis and predictive modeling.

RQ1. What are the key factors influencing cognitive strain during the coding phase of the SDLC, and how can they be quantified for effective analysis?

Justification: *Understanding the specific factors that contribute to cognitive strain is essential for developing effective interventions. By collecting and analyzing data on variables such as task complexity, work hours, stress levels, sleep quality, multi-tasking, and deadline pressure, we can quantify their impact on cognitive strain. This enables a comprehensive analysis of the mental load experienced by developers and identifies areas where improvements can be made.*

RQ2. How can the identification and management of cognitive strain improve productivity and developer well-being during the coding phase of the SDLC?

Justification: *By identifying cognitive strain through predictive models and understanding its contributing factors, we can develop strategies to manage and reduce it. This research question focuses on exploring how interventions based on data-driven insights can enhance productivity, reduce errors, prevent burnout, and improve the overall developer experience.*

1.3 Significance of the Study

This research holds significant importance for the software development industry. By identifying the key factors influencing cognitive strain and proposing practical strategies to manage it, the study contributes to improving developer productivity and well-being. The insights gained can help organizations optimize task allocation, manage workloads more effectively, and create supportive work environments that prevent burnout.

Moreover, the predictive models developed in this study serve as tools to facilitate early identification of cognitive strain. While machine learning techniques are employed, the focus remains on their application to address real-world challenges faced by developers during the coding phase.

1.4 Implications and Applications

Implications:

The findings of this research can lead to:

- Enhanced understanding of how various factors contribute to cognitive strain among developers.
- Improved strategies for workload management and task allocation, leading to increased productivity and code quality.
- Better support systems for developers, reducing the risk of burnout and promoting mental well-being.

Applications:

- **Organizational Policy Development:** Organizations can use the insights to formulate policies that balance workloads, set realistic deadlines, and support developer well-being.
- **Project Management Tools:** Integration of cognitive strain indicators into project management software can help in monitoring developer workloads and preventing overload.
- **Training and Development Programs:** Tailored programs can be developed to address identified factors contributing to cognitive strain, enhancing skills and coping mechanisms.
- **Wellness Initiatives:** Organizations can implement wellness programs that focus on stress management, sleep quality, and work-life balance, informed by the study's findings.

1.5 Ethical Considerations

When conducting this research, several ethical considerations are addressed:

- **Informed Consent:** Participants are fully informed about the purpose of the study, the data being collected, and how it will be used. Participation is voluntary, and consent is obtained prior to data collection.
- **Privacy and Confidentiality:** Personal and sensitive information is handled with strict confidentiality. Data is anonymized to protect participant identities, and secure storage methods are employed to safeguard information.
- **Data Usage:** Collected data is used solely for the purposes of this research. Participants have the right to withdraw their data at any point.
- **Non-Maleficence:** The research avoids causing harm to participants. The findings are intended to benefit developers by improving their work conditions and well-being.

1.6 Outline

The remainder of this thesis is structured as follows:

- **Background Study:** The background study provides a comprehensive overview of cognitive strain, detailing its impact on productivity, code quality, and developer well-being. It explores the factors influencing cognitive strain, such as task complexity, developer experience, and environmental interruptions, setting the stage for the research focus.
- **Related Work:** This section reviews existing literature on cognitive strain and its assessment methods, including traditional approaches like Function Point Analysis and self-assessment surveys. It also covers prior research on machine learning applications in predicting cognitive strain, identifying gaps that this study aims to address.
- **Methodology:** Details the data collection process, the design of the questionnaire, and the methods used for data analysis and model development (Random Forest, LSTM, LR, and KNN). It describes the processes for training, validating, and evaluating the models, as well as the metrics used to assess their performance.
- **Results and Analysis:** Presents the findings from the data analysis, the findings from the machine learning models, including performance metrics such as accuracy, precision, recall, and F1-score. It provides a detailed analysis of the results, comparing the effectiveness of different models in predicting cognitive strain.

- **Discussion:**The discussion interprets the results in relation to the research questions and objectives. It examines the implications of the findings for software development practices, considers the impact on productivity and developer well-being, and explores how the results address the gaps identified in the related work.
- **Conclusions and Future Work:**This section summarizes the key findings of the research, highlighting their contributions to the field of software development. It acknowledges the study's limitations and proposes directions for future research to further investigate cognitive strain and improve prediction models.

2.1 Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a systematic process used to develop software applications efficiently and effectively [8]. It consists of several distinct phases: planning, analysis, design, development, testing, deployment, and maintenance. Each phase serves a specific purpose and contributes to the successful delivery of a software product.

- **Planning:** This initial phase involves defining the project scope, objectives, and resources. It includes creating a project plan, identifying potential risks, and establishing timelines.
- **Analysis:** During the analysis phase, requirements are gathered from stakeholders to understand their needs and expectations. This information is used to define system specifications and functionality.
- **Design:** The design phase translates requirements into detailed system architecture and design specifications. This includes creating design models, user interfaces, and data structures.
- **Development:** In the development phase, the actual coding of the software takes place based on the design specifications. Developers write and integrate code to build the software.
- **Testing:** The testing phase involves systematically evaluating the software to identify and fix defects. Various testing methods, including unit testing, integration testing, and system testing, are employed to ensure quality and functionality.
- **Deployment:** Once the software passes testing, it is deployed to the production environment where it becomes available for end-users. This phase includes installation, configuration, and user training.
- **Maintenance:** The maintenance phase addresses any issues that arise post-deployment. It includes bug fixes, updates, and enhancements to ensure the software continues to meet user needs and remains operational.

Understanding the SDLC is crucial for managing cognitive strain effectively, as each phase introduces different challenges and demands on developers. The cognitive load

experienced during the development phase, in particular, can impact productivity and code quality [9]. Therefore, analyzing cognitive strain within the context of the development phase helps identify ways to improve developer efficiency and well-being throughout the software development process

2.2 Types of SDLC Methodologies

The Software Development Life Cycle (SDLC) encompasses various methodologies, each offering distinct approaches to managing and executing software projects. The choice of methodology can significantly impact the development process, project outcomes, and overall efficiency. Here are some commonly used SDLC methodologies:

- **Waterfall Model:**

The Waterfall model is a traditional, linear approach to software development. It follows a sequential process where each phase must be completed before the next one begins. The phases typically include requirements analysis, design, implementation, testing, deployment, and maintenance [10]. This model is straightforward and easy to manage but can be inflexible in accommodating changes once a phase is completed.

- **Agile Methodology:**

Agile is an iterative and incremental approach that emphasizes flexibility and customer collaboration. Projects are divided into small, manageable units called sprints or iterations, each delivering a functional piece of software. Agile methodologies, such as Scrum and Kanban, focus on adaptive planning, continuous improvement, and rapid delivery of working software. This approach allows for frequent feedback and adjustments based on changing requirements [11].

- **DevOps:**

DevOps integrates development and operations to enhance collaboration and streamline the software delivery process. It emphasizes automation, continuous integration, continuous deployment, and monitoring. The DevOps approach aims to improve efficiency and reduce the time between writing code and deploying it to production, fostering a culture of shared responsibility between development and operations teams [12].

- **Spiral Model:**

The Spiral model combines iterative development with elements of the Waterfall model [10]. It involves repeated cycles or "spirals" of planning, risk analysis, engineering, testing, and evaluation. Each spiral builds upon the previous one, allowing for incremental refinement and adaptation based on feedback and risk assessment. This model is well-suited for large, complex projects with evolving requirements.

- **V-Model (Validation and Verification Model):**

The V-Model is an extension of the Waterfall model, emphasizing the importance of validation and verification. Each development phase is paired with a

corresponding testing phase. For example, requirements are validated through corresponding acceptance testing, and design is verified through system testing. This approach ensures that errors are identified and addressed at each stage of the development process [10].

- **Incremental Model:**

The Incremental model involves developing software in small, manageable segments or increments. Each increment adds additional functionality to the existing system, allowing for partial implementation and early delivery of software [13]. This approach facilitates flexibility in accommodating changes and provides early feedback from users, enhancing the ability to address issues and adapt to evolving requirements.

- **RAD (Rapid Application Development):**

RAD focuses on rapid prototyping and iterative development to quickly deliver functional software. It involves user feedback and continuous refinement of prototypes to ensure that the final product meets user needs. RAD emphasizes speed and flexibility, making it suitable for projects with rapidly changing requirements and tight deadlines [14].

Each SDLC methodology has its strengths and weaknesses, and the choice of methodology should align with the project requirements, organizational goals, and development environment. Understanding these methodologies is crucial for selecting the most appropriate approach to manage cognitive strain and optimize the software development process.

2.3 The Significance of Cognitive Strain in SDLC

Understanding cognitive strain is essential not only for improving developer well-being but also for optimizing software development processes. When cognitive load becomes too high, it leads to a variety of negative outcomes, such as decreased focus, slower problem-solving abilities, and a higher likelihood of errors. In software development, where even small errors can cascade into significant problems, the consequences of cognitive overload can be particularly severe. Poor code quality due to strain-induced mistakes increases the time required for debugging and maintenance, thereby extending project timelines and raising costs [15].

Additionally, cognitive strain affects team dynamics. Developers who are overburdened may communicate less effectively, leading to misunderstandings, missed requirements, and fragmented teamwork. This is particularly problematic in Agile environments where constant collaboration and communication are critical to success. Cognitive strain can also lead to developer burnout, a state of physical and emotional exhaustion that reduces both the quality and quantity of work produced. Burnout not only leads to reduced job satisfaction and high turnover rates but also results in the loss of valuable knowledge and expertise within the team [16].

From an organizational perspective, unaddressed cognitive strain can severely undermine productivity and project delivery. As the software industry continues to demand faster development cycles with more complex systems, it is imperative for

organizations to adopt strategies that monitor and mitigate cognitive strain among developers. Managing cognitive load effectively can lead to better code quality, faster delivery times, and more sustainable work environments [17].

2.4 Predictive Modeling for Cognitive Strain Using Machine Learning Models

In this research, we employed four machine learning models—Random Forest (RF), Long Short-Term Memory (LSTM), Logistic Regression (LR), and K-Nearest Neighbors (KNN)—to predict cognitive strain among software developers during the coding phase of the Software Development Life Cycle (SDLC). The selection of these models was based on their unique characteristics, suitability for the nature of our data, interpretability, and their ability to capture different aspects of the predictive problem. This section provides a detailed justification for the inclusion of each model, supported by relevant literature.

2.4.1 Random Forest (RF)

Random Forest is a powerful ensemble learning technique that builds multiple decision trees during the training process. For classification tasks, it predicts the class that appears most frequently among the individual trees, while for regression tasks, it calculates the average prediction. Known for its robustness, Random Forest excels in handling high-dimensional data and complex datasets.

2.4.1.1 Justification for Selection

- **Handling Complex Interactions:** Cognitive strain is influenced by a variety of interdependent factors that often interact in nonlinear ways. Random Forest is particularly suited for capturing these intricate relationships without requiring extensive preprocessing of the data [18].
- **Reducing Overfitting Risks:** Due to its ensemble approach, Random Forest is less prone to overfitting, even when dealing with datasets that contain noisy or correlated features. This makes it a reliable choice for predictive modeling in uncertain environments [19].
- **Understanding Key Drivers:** One of the standout features of Random Forest is its ability to assess feature importance. This capability allows us to pinpoint the most significant factors contributing to cognitive strain, which is vital for designing effective, targeted interventions [20].
- **Proven Effectiveness:** Random Forest has demonstrated its effectiveness in psychological and human factors research, where it has been used to predict complex human behaviors and states. Its success in these domains reinforces its suitability for this study [21].

2.4.2 Long Short-Term Memory (LSTM)

LSTM Networks are a specialized type of Recurrent Neural Network (RNN) designed to learn long-term dependencies in sequential data. They are particularly effective for analyzing time-series data and uncovering temporal patterns, making them a powerful tool for understanding dynamic systems.

2.4.2.1 Justification for Selection

- **Capturing Temporal Dynamics:** Cognitive strain often fluctuates over time and may accumulate due to sustained mental effort. LSTM networks are well-equipped to capture these temporal dependencies, providing valuable insights into the patterns of cognitive strain [22].
- **Understanding Sequential Relationships:** Developers' cognitive states are not isolated—they are shaped by prior tasks and events. LSTMs excel at modeling these sequential dependencies, offering a deeper understanding of how cognitive strain develops and changes [23].
- **Flexibility with Input Lengths:** LSTM networks can handle input sequences of varying lengths, which is particularly useful when analyzing the diverse work patterns of developers. This flexibility ensures that the model adapts to real-world scenarios [24].
- **Applications in Similar Domains:** LSTM has shown its effectiveness in applications like mental workload assessment and stress prediction. These successes highlight its potential for accurately modeling cognitive states, including strain, in complex environments [25].

2.4.3 Logistic Regression (LR)

Logistic Regression is a straightforward and widely used statistical model that applies a logistic function to model a binary outcome. Its simplicity and interpretability make it a staple for classification tasks, especially when clarity in understanding relationships between variables is essential.

2.4.3.1 Justification for Selection

- **Interpretability:** Logistic Regression offers clear insights into how each independent variable influences the probability of cognitive strain, making it easier to understand and communicate the impact of specific factors [26].
- **Baseline Performance:** As a baseline model, Logistic Regression provides a benchmark to evaluate the effectiveness of more complex models, helping to determine if additional complexity improves predictive performance [27].
- **Efficiency and Practicality:** Its computational efficiency makes it ideal for initial analyses or scenarios with limited data, ensuring quick and reliable results [28].

- **Use in Similar Studies:** Logistic Regression has been successfully used in occupational health studies to model outcomes related to stress and mental health, demonstrating its relevance for this research [29].

2.4.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors is a simple, non-parametric learning algorithm that predicts the class of a data point based on the majority class among its nearest neighbors. Its intuitive approach makes it a versatile tool for classification and regression tasks.

2.4.4.1 Justification for Selection

- **Simplicity and Intuitiveness:** KNN is straightforward to implement and understand, making it an excellent choice for exploratory analysis and gaining quick insights into the data [30].
- **No Assumption About Data Distribution:** Unlike many other models, KNN does not rely on assumptions about the underlying data distribution, which is particularly advantageous when dealing with complex or poorly understood datasets [31].
- **Capturing Local Patterns:** KNN is effective at capturing local structures and clusters in the data, which can reveal subtle patterns indicative of cognitive strain [32].
- **Complementary Approach:** Adding KNN to the analysis offers a complementary methodology, enriching the overall evaluation of predictive capabilities and ensuring diverse insights [33].

2.4.5 Rationale for Model Selection

The choice of these four models was a deliberate strategy designed to capture a wide range of analytical perspectives and strengths:

- **Diverse Methodologies:** By including models from distinct algorithmic families—ensemble methods (Random Forest), neural networks (LSTM), statistical regression (Logistic Regression), and instance-based learning (KNN)—we aim to uncover varied patterns and relationships within the data, ensuring a comprehensive analysis.
- **Balance of Complexity and Interpretability:** Random Forest and LSTM excel at capturing complex, nonlinear interactions, while Logistic Regression and KNN provide a simpler, more interpretable lens. This balance helps translate findings into actionable insights while addressing complex data structures.
- **Suitability for Data Characteristics:** These models were chosen for their ability to handle the size, variability, and potential nonlinearities of our dataset, ensuring robust performance under diverse conditions.

2.5. Challenges of Cognitive Strain in the Development/Coding Phase of SDLC 13

- **Support from Literature:** Each model has a strong track record of success in related research areas, reinforcing their suitability for exploring cognitive strain in this context.

2.4.6 Exclusion of Other Models

While several other machine learning models were considered, they were ultimately excluded for specific reasons:

- **Advanced Deep Learning Models:** Techniques like Transformer networks, while powerful, demand large datasets and significant computational resources. Given the scope and constraints of our study, they were deemed impractical [34].
- **Support Vector Machines (SVM):** Although SVMs are highly effective for classification, they tend to be less interpretable and require meticulous parameter tuning. These factors made them less suitable for our objectives, which emphasize both performance and practical usability [35].
- **Naive Bayes:** This model relies on strong independence assumptions, which are unlikely to hold in our dataset due to potential correlations among features. As a result, it may not provide reliable predictions in this context [36].
- **Other Ensemble Methods (e.g., XGBoost):** While methods like XGBoost are known for their performance, they add complexity without offering substantial interpretability improvements over Random Forest. Any potential performance gains would likely be marginal and not justify the added effort [37].

2.5 Challenges of Cognitive Strain in the Development/Coding Phase of SDLC

Cognitive strain in the coding phase of the Software Development Life Cycle (SDLC) arises from managing complex codebases, frequent task switching, tight deadlines, and evolving requirements. Alawad et al. [38] highlight that unreadable and highly complex code demands significant cognitive effort, leading to increased errors and reduced productivity. Additionally, debugging presents a significant cognitive challenge, particularly in concurrent software, as outlined by Huang and Zhang [39]. Developers must also constantly adapt to new tools and frameworks, adding further mental strain, especially when dealing with rapid technological advancements.

Frequent context switching and multitasking further exacerbate cognitive load, as demonstrated by Czerwinski et al. [40], who found that interruptions disrupt focus and productivity. Tight schedules amplify this strain, leading to rushed code implementations and increased anxiety, as discussed by Ajmal et al. [41] in their study on scope creep. Additionally, Hu et al. [42] emphasize that prolonged cognitive effort without adequate breaks results in fatigue, impairing problem-solving ability and overall cognitive performance. Work-life balance remains a critical factor, as sustained pressure without proper recovery contributes to long-term burnout.

Psychological stress and collaboration challenges further increase cognitive strain. Miscommunication and unclear documentation force developers to expend additional effort aligning with team members and interpreting requirements, impacting efficiency. High-pressure environments often discourage breaks and self-care, intensifying stress and leading to cognitive depletion. Addressing these challenges through structured workflows, realistic deadlines, and a supportive work culture can significantly improve developer efficiency, reduce burnout, and enhance software quality.

2.6 Relevance to the Current Research

While the entire Software Development Life Cycle (SDLC) introduces challenges that contribute to cognitive strain, our study specifically focuses on the development phase, where the strain is most pronounced. The development phase, characterized by coding, debugging, and the integration of system components, places high cognitive demands on developers. This is due to several factors such as the complexity of code, the need to maintain focus during long coding sessions, and frequent context switching, all of which can overload a developer's mental capacity.

The relevance of the SDLC background to our study lies in how these challenges manifest during the development phase. As developers write and modify code, they must navigate through complex architectures, manage dependencies, and solve intricate problems, all of which heighten cognitive strain. Additionally, the coding phase often involves multitasking, such as switching between different projects or debugging and fixing bugs while adhering to tight deadlines. These activities can lead to cognitive overload, making it harder for developers to maintain focus and work efficiently.

By concentrating on the development phase, our study aims to pinpoint the exact moments when cognitive strain peaks, using machine learning models like Random Forest, LSTM, Logistic Regression, and K-Nearest Neighbors (KNN). These models are designed to predict strain based on key inputs such as task duration, code complexity, and interruptions—factors that are particularly relevant in this phase. Understanding how cognitive strain builds up during the development phase allows for targeted interventions, such as recommending breaks or adjusting workloads, to mitigate its effects, ultimately leading to improved developer performance and better code quality.

Thus, our focus on the development phase within the broader context of SDLC highlights the importance of managing cognitive strain at this critical point in the software creation process, where the balance between mental load and output efficiency can directly impact project success.

In recent years, understanding cognitive strain during the software development life cycle (SDLC) has become an increasingly important area of study due to its direct impact on developer productivity and well-being. Bläsing and Bornewasser (2021) explored the impact of task complexity and the use of informational assistance systems on mental workload, showing that the cognitive burden increases as tasks become more complex, leading to strain on mental resources. Their research highlights the need for tools and methods that reduce task complexity and provide real-time assistance, particularly during the coding phase, where developers must juggle multiple tasks simultaneously. Additionally, the study suggests that automation and improved user interfaces can play a key role in mitigating cognitive strain [43]. Kalakoski et al. (2020) also contributed to this area by investigating a workplace intervention designed to improve cognitive ergonomics. They found that cognitive strain could be significantly reduced through interventions targeting workload management and task design, thereby improving overall employee well-being. This research is particularly relevant to software development environments, where task load is often intense, and developers are required to engage in cognitively demanding activities for extended periods [44].

The integration of machine learning (ML) models to predict cognitive strain is an emerging field that holds great promise for improving cognitive load management in the SDLC. Zhao (2021) conducted a study focused on identifying “bad smells” in model-based systems engineering, which refers to indicators of poor design that are associated with increased cognitive strain. Zhao’s findings emphasize the potential for using machine learning models to predict these “bad smells” and provide early warnings to developers, allowing them to make design adjustments that reduce cognitive load [45]. Similarly, Alzayed et al. (2022) discussed the importance of top management involvement in addressing cognitive load issues. Their study argues that managerial strategies that incorporate ML-based cognitive load monitoring systems throughout the SDLC can help mitigate the negative effects of cognitive strain on developers by creating a supportive work environment that adapts based on real-time stress data [46]. These works underscore the importance of leadership in fostering environments where ML models are effectively employed to predict and manage cognitive load.

One of the significant challenges in predicting and managing cognitive load during the SDLC is the accurate measurement of cognitive strain. Gonçalves et al. (2021) undertook a comprehensive systematic mapping study to examine how cognitive load is measured in software development. They discovered that most existing approaches

rely on subjective self-assessment tools like the NASA-TLX, along with physiological metrics such as eye-tracking and EEG data. However, their research also revealed a significant gap in real-time monitoring solutions that could integrate seamlessly into the development workflow. Another challenge highlighted was the lack of standardization in measurement methods, making it difficult to compare results across studies. These findings stress the need for consistent, real-time, and non-invasive tools to measure and manage cognitive load, particularly during high-pressure tasks like coding and debugging. Such advancements align with the growing interest in predictive modeling to enhance developer productivity and well-being [47]. Abu-Shaqra (2020) tackled this issue by discussing the ethical considerations and risk management practices associated with monitoring cognitive load in complex sociotechnical systems. His work highlights the importance of transparency, privacy, and informed consent when collecting cognitive data, especially in high-stress coding environments where monitoring may feel invasive [48]. Meanwhile, Bogza et al. (2020) demonstrated the effectiveness of user-centered design in reducing cognitive strain by creating decision aids tailored to specific cognitive challenges. While their study focused on older adults with cognitive impairments, their methods could be adapted to software development contexts, where decision aids could be used to reduce mental effort and improve decision-making accuracy [49]. These insights into ethical considerations and tailored tools form the basis for creating responsible and effective cognitive load monitoring systems within the SDLC.

Recent studies have also emphasized the role of physiological data in enhancing the predictive accuracy of cognitive strain models. Xu and Jin (2022) examined how daily entrepreneurial stressors impact long-term leadership behaviors and well-being, showing that physiological indicators like heart rate variability can be strong predictors of cognitive strain [50]. Their findings suggest that physiological data could be integrated into machine learning models to more accurately predict when a developer is experiencing excessive cognitive load during the coding phase. Granek et al. (2022) further explored the role of usage analytics and end-user feedback in managing mental strain, particularly in high-performance environments like military training. Their findings can be applied to software development, where tracking developer behavior through usage analytics could provide early warnings of cognitive overload, allowing for preemptive intervention [51]. By combining physiological and behavioral data, these approaches offer promising avenues for building predictive models that not only detect but also prevent cognitive strain before it impacts productivity.

As the integration of cognitive and physiological data into predictive models becomes more sophisticated, the need for adaptive tools that manage cognitive strain throughout the SDLC grows. Dlamini et al. (2022) proposed several metrics for assessing software process quality in the late phases of the SDLC, which often correlates with increased stress and cognitive load [52]. Their work highlights the importance of early-phase interventions to manage strain, as issues that emerge later in the SDLC are often compounded by earlier mistakes or inefficiencies. This research suggests that predictive models for cognitive strain must take into account the cumulative nature of stress throughout the SDLC.

Phases such as coding and testing are particularly known for high cognitive load, making them prime candidates for the application of predictive models. Kadam et al. (2024) introduced a Software Reliability Growth Model that not only improves

code quality but also addresses developer well-being by reducing the mental effort required to identify and fix software bugs. By integrating this model with machine learning tools designed to monitor cognitive strain, developers can benefit from both improved code quality and reduced cognitive load during testing [53]. Härkönen (2020) explored ergonomic interventions designed to reduce cognitive strain during high-stress tasks in post-stroke rehabilitation, offering insights that could be adapted to software development. His findings highlight the potential for using ergonomic principles to design work environments that reduce mental effort during the most cognitively demanding phases of the SDLC [54].

In the realm of software testing, managing cognitive load is essential for ensuring quality assurance without overburdening developers. Najihi et al. (2022) compared Agile and traditional software testing methods, finding that Agile approaches tend to distribute cognitive load more evenly across the development cycle, which helps prevent the buildup of strain [55]. Their study suggests that Agile testing frameworks, when combined with machine learning models, could provide real-time feedback on cognitive load, allowing developers to make adjustments before the strain becomes detrimental. Sugiantoro et al. (2020) proposed a secure SDLC framework for e-commerce applications that incorporates considerations for cognitive strain in high-security environments. They argued that the cognitive load of managing security tasks, which often come with strict deadlines and high stakes, can be mitigated through automation and predictive tools designed to monitor stress levels [56]. These studies emphasize the importance of real-time cognitive load management, particularly during testing and security phases, where mistakes can be costly.

A comprehensive understanding of cognitive load management within various SDLC models is crucial for improving both developer productivity and well-being. Olorunshola and Ogwueleka (2022) reviewed multiple SDLC models, such as Waterfall and Agile, assessing their effectiveness in managing cognitive load. They found that models with iterative feedback loops, such as Agile, are more effective at distributing cognitive load across the development phases, whereas more rigid models like Waterfall tend to concentrate strain during specific phases, leading to burnout [14]. Hoti et al. (2023) conducted a study on mobile application development, revealing that different phases of the SDLC impose varying levels of cognitive strain depending on the type of application being developed. Their findings suggest that machine learning models designed to predict cognitive strain must be tailored to the specific characteristics of the project and its SDLC model [57].

3.1 Research Gap

Despite extensive research on cognitive strain within the Software Development Life Cycle (SDLC), there is a notable lack of studies that specifically quantify and predict cognitive strain during the coding phase using empirical data and predictive modeling. Existing literature often addresses cognitive load in general terms or focuses on other phases like testing or design, leaving a critical gap in understanding the unique cognitive challenges faced by developers during coding.

While machine learning has been applied to predict cognitive strain in other contexts—such as identifying system issues or analyzing physiological stress indi-

cators—its application to predicting cognitive strain specifically during the coding phase remains largely unexplored. Traditional methods like self-reported surveys and static code metrics fail to capture the dynamic and multifaceted nature of cognitive load experienced by developers.

This research aims to fill this gap by collecting empirical data on key factors influencing cognitive strain during coding and utilizing machine learning models—namely Random Forest, LSTM, Logistic Regression, and KNN—to develop effective predictive models. By integrating machine learning into this context, the study seeks to enhance predictive accuracy and enable proactive measures to mitigate cognitive overload, thereby improving developer well-being and productivity during the coding phase of the SDLC.

This chapter outlines the research methodologies employed to address the two primary research questions of this study. The chapter is structured into two main sections, each dedicated to one of the research questions (RQs):

The **Section 4.1** details the methodology for RQ1—What are the key factors influencing cognitive strain during the coding phase of the SDLC, and how can they be quantified for effective analysis?

A systematic approach is taken to identify and quantify the key factors influencing cognitive strain among software developers during the coding phase of the Software Development Life Cycle (SDLC). This involved a preliminary review of existing research to establish a theoretical foundation, followed by empirical data collection through a carefully designed questionnaire. We engaged experienced software developers in consultations to refine our parameters and ensure the relevance and practicality of the factors considered.

The **Section 4.2** details the methodology for RQ2—How can the identification and management of cognitive strain improve productivity and developer well-being during the coding phase of the SDLC?

This section explores methodologies for identifying and managing cognitive strain to enhance productivity and developer well-being. Building on findings from RQ1, machine learning models were developed to predict cognitive strain based on the quantified factors. Although the management strategies were not practically implemented, they were formulated using insights from the predictive models and supported by existing literature. These strategies provide evidence-based recommendations to mitigate cognitive strain and improve developer productivity and well-being.

By structuring the chapter around these two research questions, we present a clear and systematic account of the methodologies employed to investigate cognitive strain and propose strategies for its effective management within the context of the SDLC coding phase.

4.1 Methodology for RQ1

A systematic approach as depicted in the Figure ?? combining both qualitative and quantitative methods was undertaken. Our objective was to identify the key factors

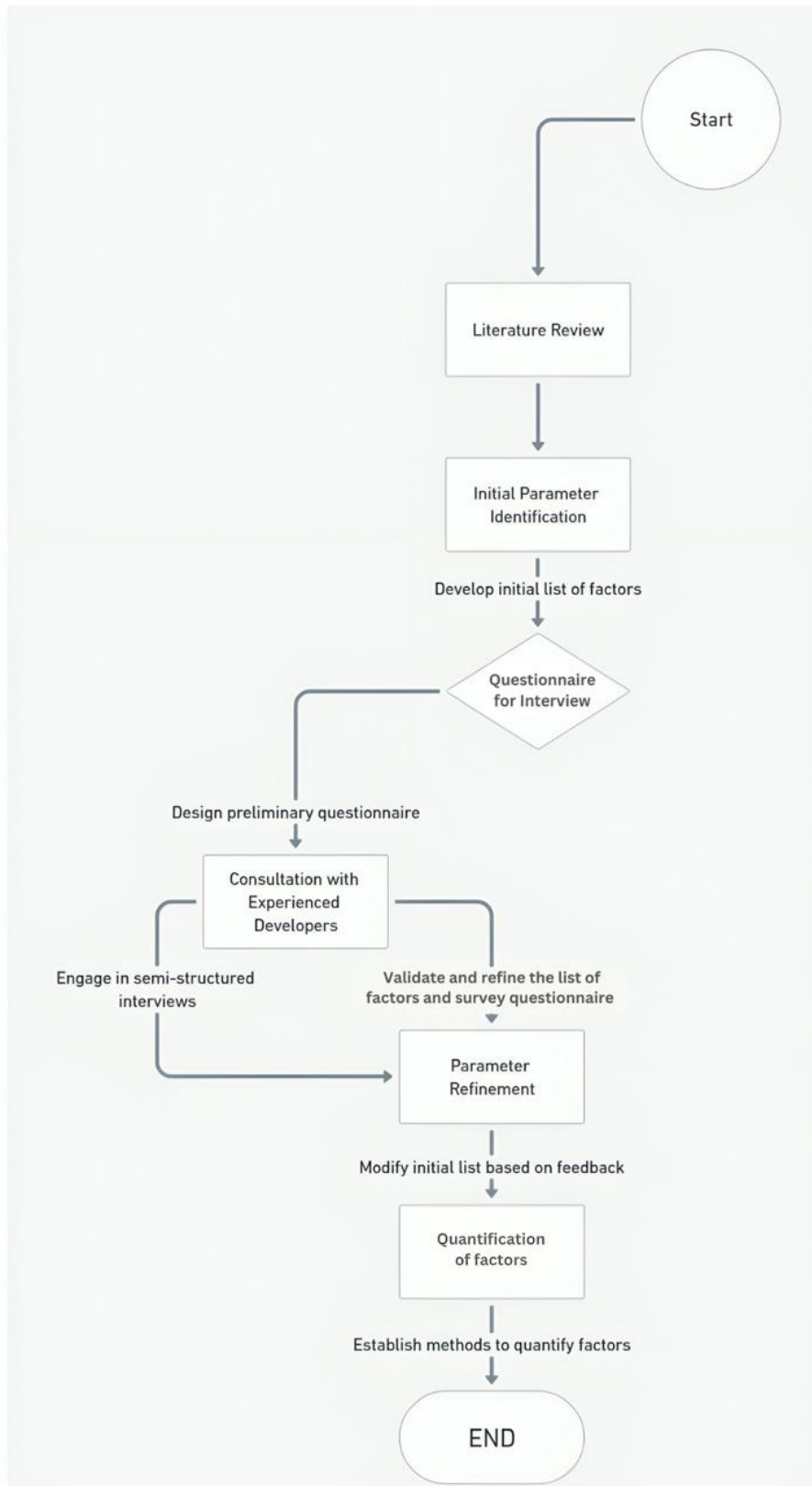


Figure 4.1: Research flow diagram for RQ1

influencing cognitive strain and establish reliable means to quantify these factors for effective analysis. The methodology involved the following key steps:

- **Literature Review:** Conducted a preliminary review of existing research to identify potential factors influencing cognitive strain in software developers.
- **Initial Parameter Identification:** Based on literature insights, we developed an initial list of factors that potentially contribute to cognitive strain during the coding phase.
- **Questionnaire Design:** Created a preliminary questionnaire aimed at collecting data on these identified factors from software developers.
- **Consultation with Experienced Developers:** Engaged in semi-structured interviews with experienced software developers to validate and refine the list of factors and the questionnaire.
- **Parameter Refinement:** Modified the initial list of factors based on developer feedback to ensure relevance, clarity, and practicality.
- **Quantification of Factors:** Established methods to quantify both objective and subjective factors, enabling effective data collection and analysis.

By systematically identifying and quantifying the key factors influencing cognitive strain, we directly addressed RQ1. This approach ensured that the factors considered were both theoretically grounded and practically relevant, facilitating effective analysis and potential application in real-world settings.

4.1.1 Preliminary Research

Objective: To Identify factors, theories, and models related to cognitive strain and Establish a theoretical foundation for parameter identification in software development.

Process:

- **Search Strategy:**
 - **Databases:** IEEE Xplore, ACM Digital Library, and Google Scholar.
 - **Time Frame:** January 2000 to December 2023.
 - **Keywords/Search Strings:** The following comprehensive search string was employed (adapted as needed for specific databases):


```
("cognitive strain" OR "cognitive load" OR "mental workload"
OR "mental strain" OR "cognitive demand" OR "cognitive fatigue")
AND
("software development" OR "software engineering" OR "SDLC"
OR "programming" OR "development team")
AND
(factor* OR model* OR theory* OR "parameter identification"
OR "predictive model" OR determinant* OR "influencing factor")
```

- **Additional Method:** A snowballing technique was applied by reviewing the reference lists and citations of the initially selected articles. This helped capture additional studies from related fields that investigate cognitive strain in high-cognitive-demand environments.
- **Selection Criteria:**
 - **Inclusion:** Peer-reviewed articles, empirical studies, and theoretical papers discussing cognitive strain or load, including those from related fields where the work context involves comparable cognitive demands.
 - **Exclusion:** Non-English articles, studies from contexts significantly different from software development, and opinion pieces lacking robust empirical or theoretical support.
- **Screening and Analysis:**
 - **Initial Screening:** Titles and abstracts were reviewed using the detailed search strings; duplicate records were removed.
 - **Full-Text Screening:** Articles that met initial criteria were examined in full, and key data regarding influencing factors, theoretical frameworks, and parameter identification methods were extracted.
 - **Synthesis:** Extracted data were compiled into a framework and mapped onto predictive models relevant to the software development context.

Outcome:

- A comprehensive list of factors, theories, and models related to cognitive strain was established.
- The inclusion of studies from related fields is justified by the similar cognitive demands, ensuring robust and transferable insights.

4.1.2 Initial Parameters and Questionnaire Design

Based on our initial understanding of factors affecting cognitive load in software development, we identified several parameters commonly associated with cognitive strain. The initial parameters included:

- **Task Complexity**
- **Work Hours**
- **Task Duration**
- **Stress Level**
- **Sleep Quality**
- **Multitasking**
- **Deadline Pressure**
- **Interruptions**

- **Code Complexity**
- **Team Communication Challenges**
- **Tool Familiarity**

We formulated a set of questions around these parameters to gather both quantitative and qualitative data related to developers' experiences during the development phase. The initial questionnaire included the following questions:

- How would you rate the complexity of your current development tasks?
- On average, how many hours do you work per day?
- What is the typical duration of a development task you handle?
- How would you describe your current stress levels related to work?
- How would you rate your sleep quality recently?
- Do you often engage in multitasking during work hours?
- Do you feel pressure from deadlines in your current projects?
- Have you experienced cognitive strain during development tasks? If so, can you describe the circumstances?

4.1.3 Consultation with Experienced Developers

We consulted with eight experienced software developers from various companies to validate the relevance of these parameters and refine our approach. The consultations were conducted individually and followed a semi-structured format:

- **Introduction**
 - Explained the purpose of the research.
 - Assured confidentiality and voluntary participation.
- **Discussion of Initial Parameters**
 - Presented the identified parameters.
 - Solicited their opinions on the relevance, clarity, and practicality of each parameter.
- **Open-Ended Questions**
 - What factors contribute most to your cognitive strain during development?
 - Are there any factors not listed that significantly affect you?
 - How do you currently manage cognitive strain or mental workload?

- What suggestions do you have for measuring cognitive strain effectively and unobtrusively?

- **Feedback on Questionnaire Design**

- Requested suggestions for improving question clarity and relevance.
- Discussed preferred methods of data collection (e.g., surveys, interviews).

- **Closing**

- Thanked them for their time and input.
- Outlined next steps and how their feedback would be utilized.

4.1.4 Feedback and Justification for Parameter Refinement

The developers provided valuable insights that guided the refinement of our parameters:

Relevance and Redundancy: Developer feedback highlighted areas where parameters could be streamlined for efficiency. For example, interruptions and multitasking were often seen as overlapping, as interruptions lead to multitasking. Combining these factors would simplify the assessment. Task complexity was also found to naturally include code complexity, as complex tasks inherently involve intricate codebases. This overlap suggested that integrating these parameters would make the study clearer and more focused.

Additionally, measuring team communication posed challenges due to its subjective nature, with varying team dynamics making it difficult to quantify. Tool familiarity was also deemed less relevant, as developers were generally proficient with their tools, and not all processes depend on specific tools. This made the tool familiarity parameter less impactful in understanding cognitive strain.

Suggestions for Streamlining Parameters: Developers emphasized the need for a concise questionnaire to reduce participant burden and improve response rates. A simpler, more focused survey would encourage higher engagement and yield more accurate data. Based on their input, several parameters were proposed for consolidation or removal. Interruptions could be merged with multitasking, and code complexity integrated within task complexity. Team communication and tool familiarity were suggested for exclusion due to their subjective nature and limited contribution to the study. These adjustments ensure the study remains relevant and efficient while respecting developers' time.

4.1.5 Justification for Removing Certain Parameters

Interruptions *Reason for Removal:* Since interruptions often result in multitasking, and given that multitasking was already a parameter, including both could lead to overlapping data. To keep the questionnaire concise and avoid redundancy, we decided to retain Multitasking as it more directly relates to personal work habits.

Code Complexity *Reason for Removal:* Developers indicated that code complexity is a component of task complexity. By asking about Task Complexity, we capture the challenges posed by both the task and the code involved. Removing Code Complexity simplifies the survey without losing critical information.

Team Communication Challenges *Reason for Removal:* While acknowledged as a factor affecting cognitive strain, developers felt it was highly subjective and influenced by many external factors, making it difficult to measure accurately in a short survey. To maintain focus on more quantifiable parameters, we chose to exclude it.

Tool Familiarity *Reason for Removal:* Most developers reported being familiar with their tools, resulting in little variance in responses. Including this parameter would add length to the survey without significantly contributing to the predictive model.

4.1.6 Finalized Parameters

Based on the feedback and the need to create an efficient and effective questionnaire, we refined our parameters to focus on the most impactful factors that could be feasibly measured. The finalized parameters for data collection were:

- **Name**
- **Company Name**
- **Task Complexity**
- **Work Hours**
- **Task Duration**
- **Stress Level**
- **Sleep Quality**
- **Multitasking**
- **Deadline Pressure**
- **Cognitive Strain**

We designed a Google Form incorporating these parameters, ensuring it was concise and user-friendly to encourage participation without imposing significant effort on respondents. The exact format of the Google Form is presented in the **Table 4.1**.

We prioritized making the Google Form accessible and quick to complete:

- **Minimal Required Fields:** Only essential questions were mandatory to respect participants' time.

Parameter	Question	Response Type	Purpose
Name	Participant's Name (Mandatory)	Text	To validate data collection and allow personalized communication while ensuring confidentiality.
Email	Participant's Email (Mandatory)	Text	For necessary follow-ups while maintaining confidentiality.
Company Name	Name of the Company (Mandatory)	Text	To provide contextual information relevant to the study.
Experience Level	What is your experience level in years?	Integer Value	To assess the participant's professional experience.
Task Complexity	How would you rate the complexity of your current development tasks?	Multiple Choice (Low, Medium, High)	To measure perceived complexity of development tasks.
Work Hours	On average, how many hours do you work per day?	Integer Value	To gather insights into participants' daily work duration.
Task Duration	How many hours does your typical development task take?	Integer Value	To measure the average duration of tasks.
Stress Level	How would you describe your current stress level related to work?	Multiple Choice (Low, Medium, High)	To evaluate stress levels among participants.
Sleep Quality	How would you describe your recent sleep quality?	Multiple Choice (Poor, Average, Good)	To understand the impact of work on sleep quality.
Multitasking	Do you often engage in multitasking during work hours?	Multiple Choice (Yes, No)	To determine the prevalence of multitasking among participants.
Deadline Pressure	How would you rate the deadline pressure in your current projects?	Multiple Choice (Low, Medium, High)	To evaluate the impact of deadlines on participants.
Cognitive Strain	Have you experienced cognitive strain during development tasks recently?	Multiple Choice (Yes, No)	To measure cognitive strain levels in participants.
Additional Comments	Please share any other relevant insights or experiences.	Text (Optional)	To collect any additional feedback or observations from participants.

Table 4.1: Questionnaire Parameters and Their Purpose

- **User-Friendly Format:** Scales and multiple-choice options simplified responses and facilitated data analysis.
- **Anonymity and Confidentiality:** Participants could choose to remain anonymous, encouraging honest and candid responses.
- **Time Efficiency:** The form was designed to be completed in under five minutes to avoid disrupting participants' workflow.

4.1.7 Justification for Quantifying Subjective Measures

Quantifying subjective experiences like Task Complexity and Stress Level is essential for analyzing their impact on cognitive strain. Developers can self-assess these parameters based on their professional experience and current work context.

Task Complexity: Task complexity can be quantified by developers through key dimensions, including the technical difficulty of algorithms, the novelty of new technologies or unfamiliar domains, and the extent of problem-solving required, such as complex debugging. Interdependencies, or how tasks affect other system components, also play a significant role. By rating task complexity on a categorical scale, developers provide a subjective yet valuable measure of mental effort, influenced by individual skills and experience, offering insights into cognitive load during development.

Stress Level: Stress levels can be self-reported by developers by reflecting on factors like workload, tight deadlines, work environment (e.g., noise and ergonomics), team dynamics, and personal challenges. Using a standardized scale allows these self-assessments to be quantified, providing meaningful analysis of how stress correlates with cognitive strain in the development process.

Multitasking and Deadline Pressure: These parameters are quantified through straightforward questions:

- **Multitasking:** A yes/no question captures whether the developer often engages in multitasking, known to increase cognitive load.
- **Deadline Pressure:** Frequency options (Never to Always) gauge how often developers feel pressured by deadlines.

Cognitive Strain as an Outcome Variable Cognitive Strain is the subjective experience of mental fatigue or overload. By directly asking developers whether they have experienced cognitive strain, we obtain a clear outcome variable. Analyzing this alongside other quantified parameters helps identify patterns and predictors of cognitive strain.

Using these self-reported metrics is justified for several reasons:

- **Direct Insight into Personal Experience:** Developers are best positioned to assess their experiences, providing data that external observations might miss.
- **Established Practice:** Self-assessment scales are widely used in psychological and occupational studies to measure subjective states like stress and workload.
- **Relevance to Cognitive Strain:** Parameters like Task Complexity and Stress Level are directly linked to cognitive load theory, which posits that increased task demands consume more cognitive resources.
- **Feasibility:** Self-reported measures are practical and minimally invasive, encouraging participation and honesty.

By collecting data on these parameters, we can apply machine learning techniques to identify significant predictors of cognitive strain. This approach allows us to develop models that can predict when developers are likely to experience cognitive overload, enabling proactive measures to mitigate its impact.

4.1.8 Data Collection

We distributed the finalized questionnaire via Google Forms to software developers across various companies.

Total Participants: 91 unique developers.

Total Responses: 184 (some developers provided multiple responses over time).

Ethical Considerations

- **Informed Consent:** Participants were informed about the study's purpose and their role.
- **Confidentiality:** Personal identifiers were used solely for validation and kept confidential.
- **Voluntary Participation:** Participation was entirely voluntary, with the option to withdraw at any time.

4.1.9 Threats to Validity for RQ1:

For RQ1, the primary concern is the reliance on self-reported data to assess subjective experiences such as stress levels, task complexity, and cognitive strain. Self-reports, while commonly used in psychological and human factors research, are inherently susceptible to biases like social desirability and recall errors. To mitigate these risks, we employed standardized survey instruments and ensured response anonymity to encourage honest feedback. However, the subjective nature of self-reported data remains a limitation, as individual perceptions of cognitive strain may not always align with objective indicators.

Another potential issue lies in the dataset used to quantify cognitive strain factors. The responses from 91 developers were used as a foundation for analysis, with data augmentation techniques applied to enhance the dataset. Although these techniques were carefully selected to preserve the original data distribution, there remains a risk that synthetic patterns introduced during augmentation may not fully capture real-world variations. This limitation underscores the need for future research to validate findings using larger, non-augmented datasets that can provide a more comprehensive representation of cognitive strain in software development.

Additionally, while our study focused on professional software developers working within typical coding environments, the findings may not generalize to all development contexts. Variations in organizational culture, project complexity, and individual work habits may influence cognitive strain differently across different settings. As such, caution is needed when applying these findings beyond the specific scope of our study.

4.2 Methodology for RQ2

We adopted a research approach as depicted in Figure 4.2 that combined machine learning model development, performance evaluation, and strategy formulation based on empirical findings and literature insights. This methodology allowed us to predict cognitive strain among developers and propose management strategies supported by existing literature. The key steps involved were:

- **Data Preparation and Augmentation:** Enhanced the collected dataset to ensure robustness for machine learning analysis.
- **Machine Learning Model Development:** Applied various machine learning algorithms to predict cognitive strain.
- **Model Evaluation and Selection:** Assessed model performance using specific metrics to identify the most effective predictive model.
- **Strategy Development:** Formulated strategies for managing cognitive strain based on model insights and supported by existing literature.
- **Integration of Findings:** Aligned the strategies with theoretical frameworks and best practices identified in the literature to improve productivity and well-being.

4.2.1 Data Preparation and Augmentation

To prepare the dataset for effective machine learning analysis, we implemented data preprocessing steps, including data augmentation and transformation techniques. The initial dataset consisted of 184 responses collected through a Google Form from 91 unique developers. Some developers provided multiple responses, resulting in a total of 184 entries. However, this dataset size was relatively small for training robust machine learning models such as Random Forest, Long Short-Term Memory networks (LSTM), Logistic Regression (LR), and K-Nearest Neighbors (KNN), which

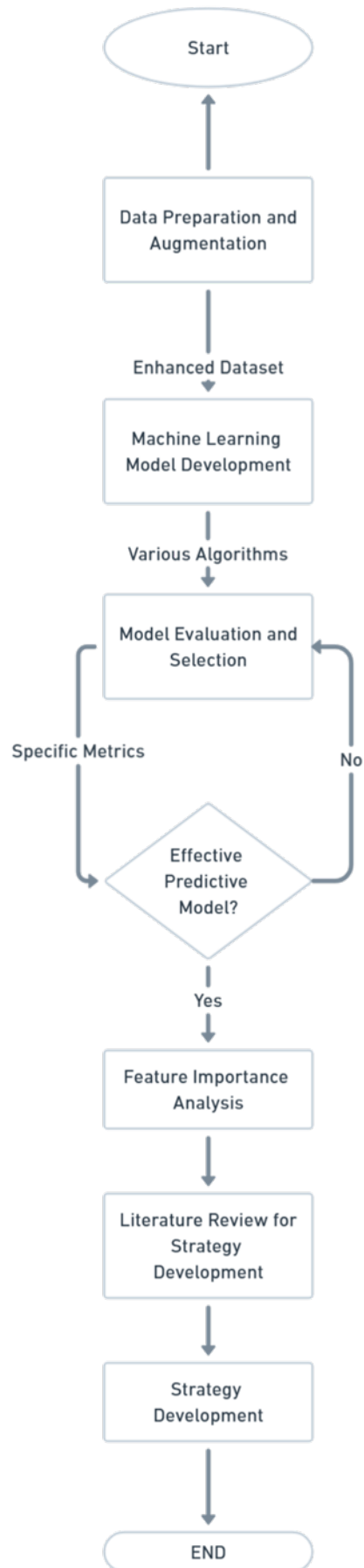


Figure 4.2: Research flow diagram for RQ2

generally perform better with larger datasets.

To address this limitation, we applied data augmentation methods to synthetically expand the dataset. Data augmentation involved generating additional data samples based on the existing responses while maintaining the underlying patterns and distributions. This process increased the dataset size to 9,863 instances, providing a more substantial foundation for training and validating our machine learning models.

The data preprocessing steps included:

- **Data Augmentation:** Techniques like synthetic data generation and noise injection were applied to create new data points. To address class imbalances, we used SMOTE (Synthetic Minority Over-sampling Technique), ensuring a balanced class distribution where needed.
- **Data Cleaning and Encoding:** We performed standard cleaning procedures, such as handling missing values and encoding categorical variables into numerical formats. Numerical features were normalized or standardized, ensuring all variables contributed equally to model training.
- **Dataset Splitting:** The augmented dataset was divided into training and testing sets, with 80% allocated for training and 20% for testing. This split allowed us to evaluate model performance reliably on unseen data.

Data augmentation was essential for several reasons:

- **Improving Model Generalization:** A larger, more diverse dataset enabled the models to generalize more effectively, improving their ability to perform well on unseen data.
- **Reduced Overfitting:** By increasing the dataset size, we minimized the risk of the models overfitting to the training data, promoting more reliable predictions.
- **Enhancing Class Representation:** Augmentation techniques like SMOTE ensured balanced class distributions, particularly crucial for models sensitive to class imbalance.

Through these preprocessing steps, we prepared a robust dataset that supported the effective training of our machine learning models. This rigorous preparation enhanced the models' ability to learn from the data, ultimately improving their predictive accuracy in identifying cognitive strain among software developers.

4.2.2 Apply Machine Learning Models

4.2.2.1 Random Forest

The Random Forest model is an ensemble learning technique that consists of multiple decision trees. After data preprocessing, the dataset is split into training and testing

sets. The Random Forest model is trained using features such as task complexity, work hours, stress level, and deadline pressure. Each decision tree in the ensemble analyzes a different subset of data to improve the overall predictive performance. After training, the model is evaluated on the test set to assess its ability to predict cognitive strain. This approach benefits from Random Forest's ability to handle high-dimensional data and complex feature interactions.

4.2.2.2 Long Short-Term Memory (LSTM)

For LSTM, a sequential deep learning model, we preprocess the data with time-series formatting to capture the temporal nature of cognitive strain over time. Input data, such as historical patterns in task duration, work hours, and stress levels, is segmented into sequences. LSTM is particularly useful for predicting cognitive strain as it considers how these factors evolve throughout the coding phase. After training the LSTM model, we evaluate it on the test set, where its ability to capture long-term dependencies allows it to predict future cognitive strain based on past data.

4.2.2.3 Logistic Regression (LR)

For Logistic Regression, a classification model, we input the preprocessed features directly without requiring any temporal dependencies. LR is used to predict whether a developer is likely to experience low, medium, or high cognitive strain based on the independent variables like task complexity, errors, work hours, and deadline pressure. While simpler than Random Forest and LSTM, LR provides interpretable results, offering insights into how each feature impacts cognitive strain prediction.

4.2.2.4 K-Nearest Neighbors (KNN)

For K-Nearest Neighbors (KNN), we normalize all features to ensure that distance-based comparisons are meaningful. KNN works by comparing the current developer's situation (e.g., task complexity, work hours, and stress levels) to past instances in the dataset and identifying the nearest neighbors with similar conditions. The predicted cognitive strain level is determined based on the most common outcome among the closest matches.

This multi-model approach leverages the strengths of each machine learning technique to provide comprehensive predictions of cognitive strain in software developers, enabling proactive measures to support their performance and mental health.

4.2.3 Performance Metrics

Each model, Random Forest, LSTM, LR, and KNN is evaluated using the test set to assess its performance. Classification metrics such as accuracy, precision, recall, and F1-score are used to compare the models. These metrics help us determine which model is most effective in predicting cognitive strain, allowing for practical insights into managing developer workloads and improving productivity and well-being during the coding phase of the SDLC.

- **Accuracy:** This metric measures the overall correctness of the model by calculating the proportion of correctly predicted instances (both positive and negative) out of the total instances. It provides a general indication of how well the model performs across all classes.
- **Precision:** Precision focuses on the model's ability to correctly identify positive instances. It is calculated as the ratio of true positive predictions to the sum of true positive and false positive predictions. High precision indicates that the model has a low false positive rate, meaning it does not frequently misclassify negative instances as positive.
- **Recall:** Recall, also known as sensitivity or true positive rate, measures the model's ability to identify all relevant positive instances. It is calculated as the ratio of true positive predictions to the sum of true positive and false negative predictions. High recall indicates that the model successfully captures most of the actual positive instances.

4.2.4 Integration of Model Insights

Once the most effective model (Random Forest) was identified, we extracted insights to determine the key factors influencing cognitive strain:

Feature Importance Analysis:

- Used the `feature_importances_` attribute of the Random Forest model to rank features by their contribution to the model's predictions.
- Selected features with importance scores above a predefined threshold which is the mean of those importance score).

The analysis identified the most impactful factors influencing cognitive strain, including task complexity, stress level, sleep quality, and deadline pressure.

4.2.5 Literature Review for Strategy Development

Objective: To Identify and evaluate existing strategies for managing cognitive strain and enhancing well-being followed by Aligning the identified strategies with the factors from the predictive models.

Process:

- **Search Strategy:**
 - **Databases:** IEEE Xplore, ACM Digital Library, and Google Scholar.
 - **Time Frame:** 2010 to 2023.
 - **Keywords/Search Strings:** The search strategy incorporated an extended set of keywords as follows:


```
("cognitive strain management" OR "cognitive load management"
OR "mental workload management" OR "stress reduction"
OR "stress management" OR "well-being enhancement"
OR "developer wellness" OR "developer well-being"
```

```

OR "burnout prevention" OR "workload management"
OR "time management")
AND
("software development" OR "software engineering" OR "SDLC"
OR "programming")
AND
(intervention OR strategy OR technique OR "best practice"
OR "organizational intervention" OR "employee well-being"
OR "health promotion" OR "productivity enhancement")

```

- **Additional Method:** A snowballing technique was utilized by reviewing reference lists and citations of the initially selected articles. This approach was essential to capture further studies from both the software development domain and related fields with similar cognitive demands.

- **Selection Criteria:**

- **Inclusion:** Peer-reviewed articles, empirical studies, and case studies focusing on interventions and strategies for managing cognitive strain. Studies from related fields were included if their work context was analogous to that of software development.
- **Exclusion:** Non-English articles and studies not pertinent to software development or comparable high-cognitive-demand environments.

- **Screening and Analysis:**

1. **Initial Screening:** Titles and abstracts were reviewed using the detailed search strings; duplicate records were removed.
2. **Full-Text Screening:** Articles that passed the initial screening were examined in full, with data on interventions, implementation methods, and outcomes being extracted.
3. **Synthesis:** The extracted interventions and best practices were mapped to the cognitive strain factors identified earlier, resulting in evidence-based strategies.

Outcome:

- Proposed evidence-based strategies to mitigate cognitive strain and enhance developer well-being.
- Strategies include workload management, stress reduction techniques, and the promotion of healthy sleep habits.
- The inclusion of studies from related fields is justified by the similar cognitive demands, ensuring that the insights are robust and transferable.

4.2.6 Threats to Validity for RQ2:

For RQ2, one major limitation is the reliance on machine learning models to predict cognitive strain and inform management strategies. While models such as Random Forest, LSTM neural networks, Logistic Regression, and K-Nearest Neighbors were chosen for their ability to capture patterns in the data, no model is perfect. The

accuracy of predictions depends on the quality and completeness of the dataset, and potential biases in data collection may influence model performance. Future work should explore alternative or hybrid modeling approaches to improve predictive robustness.

Additionally, the study did not implement cognitive strain management strategies in a real-world setting but rather formulated them based on predictive model insights and existing literature. While these recommendations are grounded in evidence, their practical effectiveness remains untested. Future research could incorporate experimental interventions in workplace environments to evaluate the actual impact of the proposed strategies on developer well-being and productivity.

Another challenge pertains to generalizability. The strategies proposed for mitigating cognitive strain are tailored to software development environments similar to those studied, where cognitive load is high due to task complexity, debugging, and frequent context switching. However, different teams, industries, or programming paradigms may require customized interventions. This limitation suggests the need for further studies examining cognitive strain across diverse software development contexts.

Lastly, while the Random Forest algorithm was selected for its interpretability in feature importance analysis, relying on a single analytical method may constrain the breadth of insights. Although the model performed well in identifying key predictors of cognitive strain, complementary approaches—such as deep learning models or human-in-the-loop evaluations—could reveal additional patterns and improve intervention strategies.

In this section, we present the findings from our study on cognitive strain among code developers. Drawing from 184 responses provided by 91 unique developers, we analyzed key factors contributing to cognitive strain during the software development lifecycle. These factors include stress levels, sleep quality, deadline pressure, and task complexity. Through descriptive statistics and data visualizations, we highlight the distribution and impact of each factor on developers' cognitive performance and overall well-being. Furthermore, we present the results of machine learning models applied to this data, demonstrating their effectiveness in predicting cognitive strain. This comprehensive analysis provides valuable insights into the challenges faced by developers and emphasizes the significance of accurate prediction models in understanding cognitive strain within software development environments.

5.1 Results of RQ1

5.1.1 Results of preliminary research for Factors, Theories, and Models

This section presents a preliminary investigation into factors that may influence cognitive load and productivity in software development. The factors listed have been identified based on an initial review of relevant literature and are intended to provide a starting point for further exploration. These factors encompass elements related to task complexity, environmental influences, and individual cognitive conditions.

Table 5.1 summarizes these factors, providing a brief description and references to supporting studies. This table serves as an initial framework for understanding potential contributors to cognitive demands in software development.

Factors	Description	Supporting Literature
Task Complexity	The degree of intricacy and difficulty inherent in a coding task that requires significant cognitive processing.	[3]; [58]; [59]; [60]

Factors	Description	Supporting Literature
Work Hours	The total amount of time spent on work, with extended hours potentially leading to mental fatigue and reduced cognitive performance.	[61]; [62]; [63]; [64]; [65]; [66]; [67]
Task Duration	The length of time needed to complete a task, which can influence developers' sustained attention and overall cognitive load.	[68]; [69]; [70]; [71]; [72]
Stress Level	The psychological pressure experienced during work, impacting concentration and decision-making abilities.	[73]; [74]; [75]; [76]; [77]; [78]; [79]; [80]; [81]
Sleep Quality	The effectiveness of restorative sleep, which is critical for maintaining optimal cognitive functions and alertness.	[82]; [83]; [84]; [85]; [86]; [87]; [88]; [89]; [90]
Multitasking	The simultaneous handling of multiple tasks that can split attention and increase mental strain.	[91]; [92]; [93]; [94]; [95]; [96]; [97]; [98]; [99]
Deadline Pressure	The time constraints imposed on project completion that elevate stress and intensify cognitive demands.	[100]; [101]; [102]; [103]; [104]; [105]; [106]; [107]
Interruptions	Unplanned disruptions during work that break concentration and contribute to higher cognitive load.	[108]; [109]; [110]; [111]; [112]; [113]; [114]; [115]
Code Complexity	The intricacy of code structure and logic that can challenge a developer's understanding and cognitive processing.	[15]; [116]; [117]; [118]; [5]; [47]
Team Communication Challenges	Difficulties in collaborative exchanges that may lead to misunderstandings and increased cognitive effort to coordinate work.	[119]; [1]; [120]; [121]; [122]; [123]; [124]
Tool Familiarity	The level of ease and proficiency with which developers can use their software tools, influencing cognitive efficiency and productivity.	[125]; [15]; [126]; [127]; [9]

Table 5.1: Factors identified from Literature Review

5.1.2 Finalized Key Factors Influencing Cognitive Strain

Through a combination of factors identified from the literature review and developer consultations, the following seven key factors were identified as the most significant contributors to cognitive strain.

1. Task Complexity
2. Stress Level
3. Sleep Quality
4. Deadline Pressure
5. Multitasking
6. Task Duration
7. Work Hours

These factors form the foundation for understanding and quantifying cognitive strain in software development contexts.

5.1.3 Cognitive Strain Distribution Across Factors

The cognitive strain distribution was analyzed across both categorical and numeric variables, as summarized in Tables 5.2 and 5.3.

5.1.3.1 Categorical Variables

Factor	Category	High Strain (%)	Medium Strain (%)	Low Strain (%)
Task Complexity	High	37.50	28.57	33.93
	Medium	33.78	37.84	28.38
	Low	33.33	31.48	35.19
Stress Level	High	40.54	35.14	24.32
	Medium	30.56	26.39	43.06
	Low	36.00	38.67	25.33
Sleep Quality	Poor	33.33	41.67	25.00
	Average	35.16	30.77	34.07
	Good	35.09	31.58	33.33
Deadline Pressure	High	35.71	19.64	44.64
	Medium	34.25	41.10	24.66
	Low	34.55	36.36	29.09

Table 5.2: Cognitive Strain Distribution Across Factors

Table 5.2 highlights the distribution of cognitive strain levels across categorical factors, including **Task Complexity**, **Stress Level**, **Sleep Quality**, and **Deadline Pressure**. Key observations include:

- **Task Complexity:** High task complexity shows the highest percentage of **High Strain** (37.50%), indicating the mental demands of complex tasks.
- **Stress Level:** Elevated stress levels are strongly associated with **High Strain** (40.54%), emphasizing workplace pressure’s impact on cognitive load.
- **Deadline Pressure:** High deadline pressure contributes to increased **High Strain** (35.71%) but also shows a significant proportion of **Low Strain** (44.64%), reflecting the nuanced effects of time constraints.
- **Sleep Quality:** Poor sleep quality correlates with higher strain levels, reinforcing the importance of personal well-being in managing mental fatigue.

5.1.3.2 Numeric Variables

	Mean	Median	Std. Dev.	Min	Max
Work Hours	7.24	7.50	1.66	4.00	10.00
Task Duration	4.55	5.00	1.67	1.00	7.50
Experience Level	3.34	2.00	3.25	1.00	12.00

Table 5.3: Cognitive Strain Distribution Across Numeric Variables

Table 5.3 provides a statistical summary of numeric variables, including **Work Hours**, **Task Duration**, and **Experience Level**. Key patterns include:

- **Work Hours:** The average work hours (7.24 hours/day) suggest a consistent workload, with extended hours potentially exacerbating cognitive strain when paired with other stressors.
- **Task Duration:** An average task duration of 4.55 hours indicates prolonged focus requirements, contributing to cognitive strain in high-demand scenarios.
- **Experience Level:** The diverse range of experience levels (average: 3.34 years, max: 12 years) highlights varying abilities to manage strain, with less experienced developers potentially facing greater challenges.

5.1.4 Patterns and Insights

The analysis reveals several critical patterns and interactions among the identified factors:

- **Compounding Effects:** High **Task Complexity** and **Stress Level** jointly contribute to increased cognitive strain, reflecting the interplay between task demands and environmental pressures.
- **Moderate Balance:** Moderate **Stress Levels** and **Deadline Pressure** exhibit balanced strain distributions, indicating optimal productivity without excessive mental fatigue.

- **Impact of Well-Being: Poor Sleep Quality** consistently correlates with higher strain levels, emphasizing the role of personal well-being in mitigating cognitive load.

5.2 Results for RQ2

This section discusses the outcomes of machine learning analyses aimed at predicting cognitive strain, alongside insights derived from feature importance analysis. These findings informed the development of strategies for managing cognitive strain, supported by existing literature.

5.2.1 Machine Learning Model Performance

To predict cognitive strain, four machine learning models were developed and evaluated: Random Forest, Long Short-Term Memory (LSTM), Logistic Regression (LR), and K-Nearest Neighbors (KNN). The performance metrics for each model are summarized in Table 5.4.

Model	Accuracy	Precision	Recall	F1 Score
Random Forest (RF)	0.991	0.991	0.991	0.991
Logistic Regression (LR)	0.616	0.625	0.616	0.618
LSTM	0.808	0.822	0.808	0.803
K-Nearest Neighbors (KNN)	0.625	0.626	0.616	0.618

Table 5.4: Performance Metrics of Different Models

Table 5.4 summarizes the performance metrics for the models used to predict cognitive strain during the coding phase of the SDLC. The metrics evaluated include Accuracy, Precision, Recall, and F1 Score.

5.2.2 Performance Comparison

The comparative analysis reveals significant differences in model performance:

- **Random Forest (RF):** Random Forest emerged as the top-performing model, achieving near-perfect scores across all metrics, with values close to 0.991. This indicates exceptional predictive capabilities and an impressive balance between precision and recall, making it highly reliable for this task.
- **LSTM:** LSTM networks performed moderately well, with an accuracy of 0.808 and a precision of 0.822. These results demonstrate its ability to reasonably balance identifying positive cases and minimizing false positives, making it a viable option for prediction.
- **Logistic Regression (LR) and KNN:** Both Logistic Regression and K-Nearest Neighbors delivered similar performance, with accuracies hovering around 0.62. These results suggest that these models are less suited for this predictive task, as they lack the reliability and precision exhibited by RF and LSTM.

5.2.3 Overall Insights

Random Forest stands out as the most effective model for predicting cognitive strain in this context, outperforming the others by a significant margin. While LSTM does not match RF's accuracy, it still provides acceptable performance and could be considered a secondary option. However, the comparatively lower accuracy and F1 scores of Logistic Regression and KNN indicate that these models may not be adequate for this application.

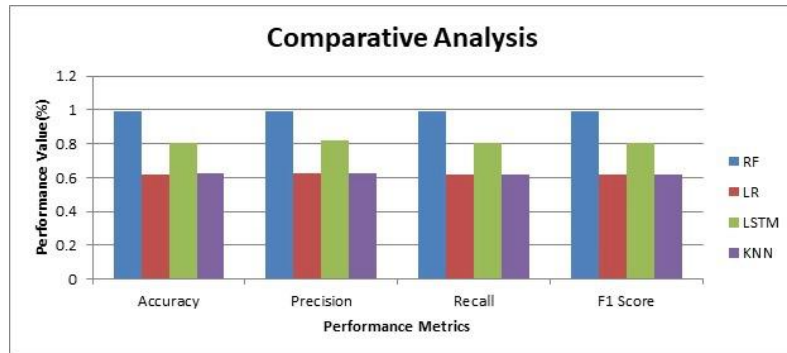


Figure 5.1: Comparative Analysis of Machine Learning Models

Figure 5.1 visually compares the performance metrics of the four models. Each model's performance values are represented as percentages, with the models color-coded for clarity.

5.3 Confusion Matrix Analysis

To further evaluate the models, we examine their confusion matrices, which provide detailed insights into the classification performance.

5.3.1 Random Forest

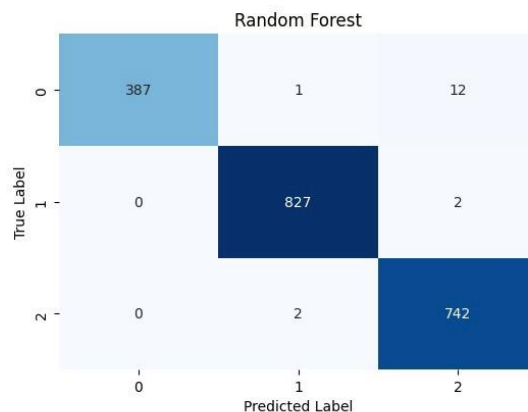


Figure 5.2: Confusion Matrix of Random Forest Model

Figure 5.2 shows the confusion matrix for the Random Forest model:

- **Class 0:** 387 true positives, few misclassifications.
- **Class 1:** 827 true positives, minimal misclassifications.
- **Class 2:** 742 true positives, very few misclassifications.

Interpretation: The Random Forest model demonstrates excellent classification accuracy across all classes, with negligible misclassifications.

5.3.2 Logistic Regression

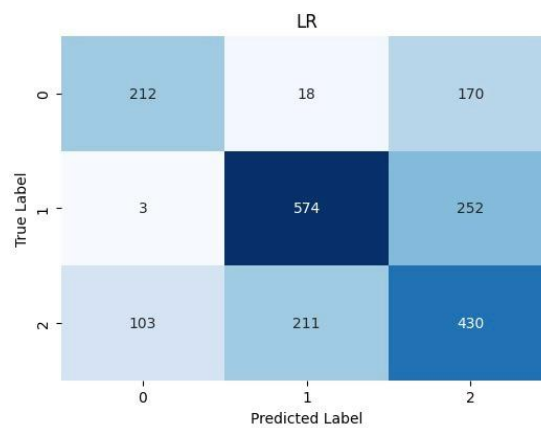


Figure 5.3: Confusion Matrix of Logistic Regression Model

Figure 5.3 presents the confusion matrix for the Logistic Regression model:

- **Class 0:** Moderate true positives, significant misclassifications, especially into Class 2.
- **Class 1:** Reasonable true positives but with misclassifications into Class 2.
- **Class 2:** Lower true positives, with considerable misclassifications into Classes 0 and 1.

Interpretation: The Logistic Regression model struggles to accurately classify Classes 0 and 2, leading to a higher rate of misclassifications.

5.3.3 LSTM

Figure 5.4 depicts the confusion matrix for the LSTM model. While specific values are not provided, the general observations are:

- **Class 0:** Good number of true positives with some misclassifications.
- **Class 1:** High true positives, indicating strong performance in this class.

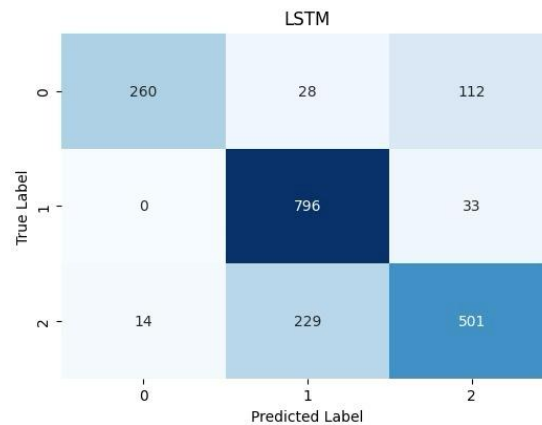


Figure 5.4: Confusion Matrix of LSTM Model

- **Class 2:** Moderate true positives with misclassifications into other classes.

Interpretation: The LSTM model performs better than Logistic Regression but does not reach the accuracy levels of Random Forest, particularly in distinguishing Class 2.

5.3.4 K-Nearest Neighbors (KNN)

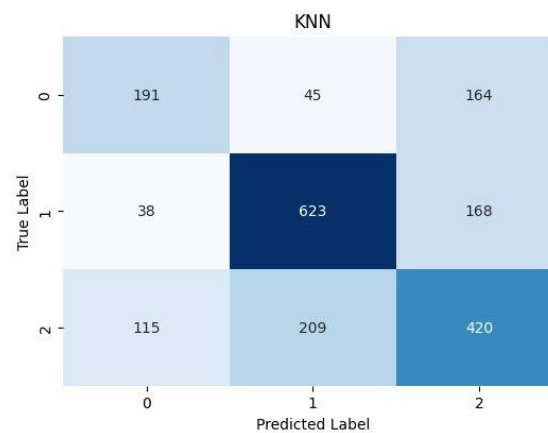


Figure 5.5: Confusion Matrix of KNN Model

Figure 5.5 illustrates the confusion matrix for the KNN model:

- **Class 0:** Lower true positives, with misclassifications into Classes 1 and 2.
- **Class 1:** Highest true positives among the classes, but still with notable misclassifications.
- **Class 2:** Significant misclassifications, indicating difficulty in correctly predicting this class.

Interpretation: The KNN model has limited effectiveness, particularly in distinguishing between Classes 0 and 2.

5.3.5 Summary of Confusion Matrix Analyses

The confusion matrices highlight the following:

- **Random Forest:** Superior performance with high true positives and minimal misclassifications across all classes.
- **LSTM:** Reasonable performance but with some confusion between classes, particularly Class 2.
- **Logistic Regression and KNN:** Struggle with accurate classification, leading to higher misclassification rates and lower reliability.

The Random Forest model outperformed the others across all metrics, making it the most effective model for predicting cognitive strain.

5.3.6 Feature Importance Analysis

The Random Forest model's `feature_importances_` attribute was used to rank the predictors of cognitive strain. The analysis identified the following key factors, which informed the proposed strategies and interventions:

- **Task Complexity** (importance score: 0.35): The most significant contributor to cognitive strain. Complex tasks often exceeded individual expertise, leading to errors and delays. This underscores the need for skill-based task allocation and training programs.
- **Stress Level** (importance score: 0.28): Elevated stress levels were strongly correlated with cognitive strain. High workloads, tight deadlines, and multitasking were primary contributors, highlighting the importance of stress management resources and workload balancing.
- **Deadline Pressure** (importance score: 0.22): Unrealistic deadlines increased the likelihood of cognitive strain, suggesting that collaborative and flexible deadline-setting practices are crucial.
- **Sleep Quality** (importance score: 0.15): Developers reporting poor sleep quality frequently experienced cognitive strain, emphasizing the need for flexible scheduling and promoting rest.
- **Multitasking** (importance score: 0.12): Frequent task switching divided attention and led to cognitive overload. This finding aligns with the proposed strategies to reduce multitasking and manage interruptions.
- **Work Hours** (importance score: 0.10): Extended work hours correlated with higher cognitive strain, reinforcing the need for balanced workloads and adherence to reasonable work hours.

The feature importance analysis revealed a comprehensive set of factors influencing cognitive strain, with Task Complexity, Stress Level, Deadline Pressure, Sleep Quality, Multitasking, and Work Hours emerging as the most significant. These findings validated the predictive power of the Random Forest model and provided actionable insights for intervention development.

5.3.7 Strategies for Managing Cognitive Load in Software Development

Building upon the predictive model developed to identify cognitive strain among developers, this section explores strategies that have been suggested in the literature to mitigate their effects. These strategies focus on balancing workloads, optimizing task assignments, and improving overall well-being to enhance developer productivity and mental health.

Table 5.6 summarizes key strategies derived from preliminary literature, outlining their descriptions, possible interventions, and supporting references. This table serves as an initial framework for understanding how various approaches can be employed to manage cognitive demands in software development environments.

Strategies	Description	Interventions	Supporting Literature
Workload Balancing	Prevents cognitive strain caused by excessive work hours.	Adjust workloads evenly. Encourage adherence to standard work hours.	[128]; [129]; [130]
Task Allocation According to Expertise	Aligns task complexity with developer skill levels.	Assign tasks based on expertise. Provide training for skill development.	[131]; [132]; [130]
Flexible Scheduling	Addresses cognitive strain caused by poor sleep and rigid work hours.	Implement flexible work hours. Promote rest and recovery.	[133]; [134]
Minimizing Multi-tasking	Reduces cognitive strain from frequent task-switching.	Encourage single-task focus. Manage interruptions.	[135]; [136]; [137]
Realistic Deadline Setting	Prevents stress from unrealistic time pressures.	Plan deadlines collaboratively. Allocate buffer time.	[138]; [139]; [140]; [141]
Providing Stress Management Resources	Supports mental well-being and reduces chronic stress.	Offer wellness programs and stress management workshops. Promote a supportive work culture.	[142]; [143]; [144]; [145]

Table 5.6: Strategies identified from Literature Review

The subsequent discussion chapter builds on these results, proposing targeted strategies that address the root causes of cognitive strain. Each strategy is grounded in both the findings of this study and established theoretical frameworks, ensuring relevance and practical applicability. By connecting these data-driven insights to actionable recommendations, the discussion emphasizes how organizations can mitigate cognitive strain, improve productivity, and enhance developer well-being.

The coding phase of the Software Development Life Cycle (SDLC) is a critical period where developers face high cognitive demands. Our study confirms that cognitive strain is not the result of a single factor, but emerges from a network of interacting influences. In particular, high task complexity, extended work hours, stringent deadline pressures, poor sleep quality, elevated stress levels, frequent multitasking, and prolonged task durations jointly contribute to cognitive overload. These factors interact dynamically, overwhelming the limited capacity of working memory and leading to degraded performance, increased error rates, and reduced overall well-being.

6.1 Interpretation of Key Findings

The study identified seven primary factors influencing cognitive strain:

1. **Task Complexity**
2. **Work Hours**
3. **Stress Level**
4. **Sleep Quality**
5. **Multitasking**
6. **Deadline Pressure**
7. **Task Duration**

Interconnected Nature of Influencing Factors

One of the most salient insights is the interconnectedness of these factors. Rather than operating in isolation, each element compounds the others, resulting in a cumulative burden that strains developers' cognitive resources. For example:

- **Task Complexity and Deadline Pressure:** High task complexity, especially when combined with tight deadlines, forces developers to process intricate code logic while simultaneously managing time constraints. This dual demand pushes the working memory beyond its optimal capacity, making it harder for developers to maintain accuracy and consistency in their work. The literature on cognitive load theory suggests that such conditions not only impair problem solving but also limit the capacity for creative and strategic thinking.

- **Work Hours and Sleep Quality:** Extended work hours often erode the time available for restorative sleep. The resulting sleep deprivation detrimentally affects critical cognitive functions—including attention, memory consolidation, and problem-solving ability—which are essential for coding tasks. This creates a vicious cycle: poor sleep quality further reduces cognitive performance, which may then necessitate longer work hours to complete tasks, exacerbating the strain.
- **Multitasking and Stress Level:** Frequent task switching inherent in multitasking forces developers to constantly reorient their focus. The mental effort required to shift between different cognitive contexts increases overall stress levels. This heightened stress not only impairs cognitive functions such as working memory and decision-making but also accelerates the onset of burnout.

Theoretical Implications

Our findings are grounded in established theoretical frameworks. Cognitive load theory, as introduced by Sweller (1988) [132], posits that working memory is limited and that an overload—whether from intrinsic task difficulty or extraneous distractions—can impair learning and performance. In our context, coding tasks that require simultaneous processing of complex algorithms and managing rapid deadlines closely mirror conditions of high intrinsic and extraneous cognitive loads.

Additionally, the Job Demands-Resources model [146] offers a useful lens through which to interpret our findings. This model explains that when job demands (e.g., complex tasks, long work hours) exceed the available resources (e.g., adequate rest, supportive work environment), employees experience strain that negatively affects their performance and health. Our results validate this theory by demonstrating that a sustained imbalance between demands and resources leads to cognitive overload and burnout.

A further theoretical contribution is the successful application of predictive modeling using a Random Forest classifier. The model's high accuracy not only confirms the selection of the key predictors but also illustrates that these factors interact in systematic, quantifiable ways. This empirical evidence moves our understanding beyond theoretical postulates and establishes a concrete foundation for subsequent research and intervention design.

Practical Implications

The practical implications of our study are multifold and suggest that any effective intervention must be holistic. Key recommendations include:

Holistic Workload Management

- **Integrate Task Assignment and Scheduling:** Aligning tasks with the appropriate level of developer expertise can reduce unnecessary cognitive strain. For example, complex tasks should be assigned to developers with the requisite skills, while routine tasks might be delegated to those with less experience.

Additionally, setting realistic deadlines helps minimize the compounded effects of high complexity under time pressure.

- **Flexible Work Arrangements:** Flexible scheduling allows developers to work at times that suit their personal productivity rhythms. Such flexibility not only supports adequate sleep but also enables dedicated “focus periods” that protect against interruptions.

Optimized Work Environment Design

- **Minimize Interruptions:** Designing workspaces with dedicated quiet zones or “focus rooms” can help reduce the cognitive costs associated with frequent task switching. This measure is particularly relevant in open office environments, where distractions are common.
- **Supportive Technological Tools:** Leveraging advanced Integrated Development Environments (IDEs) and collaboration platforms that provide cognitive support—such as error-mediated integration or interactive code visualization—can help developers manage cognitive load more effectively.

Personalized Support and Development

- **Mentorship and Training Programs:** Tailoring support to individual differences is critical. Novice developers, in particular, may benefit from structured mentorship and targeted training to build robust mental schemas. In contrast, experienced developers might require strategies to prevent the negative effects of over-guidance—a phenomenon known as the expertise reversal effect.
- **Stress Management Initiatives:** Programs that offer stress relief, mental health resources, and mindfulness training can empower developers to build resilience. By addressing stress proactively, organizations can prevent its escalation into chronic cognitive overload.

Reflections on Empirical Methodology and Limitations

Although our study relied in part on self-reported data and the use of augmented datasets—which inherently introduces some limitations—the consistency of observed patterns provides strong support for our conclusions. It is important to note that while self-reported measures may be susceptible to biases such as social desirability, the integration of predictive modeling has helped validate the key factors in a quantifiable manner. Furthermore, our study captures only a snapshot in time; future longitudinal research is needed to observe how cognitive strain evolves over different phases of projects and under varying organizational conditions.

Alternative Perspectives and Broader Considerations

It is also worth considering that not all cognitive load is inherently negative. A moderate level of cognitive strain, often referred to as *eustress* [147], can enhance

alertness and drive innovation. However, when cognitive demands surpass a critical threshold, they lead to detrimental outcomes such as increased error rates, reduced creativity, and burnout. Additionally, cultural norms within organizations—such as the valorization of long work hours or multitasking—can influence how cognitive strain is perceived and reported. Recognizing these nuances is essential for tailoring interventions that not only address the symptoms but also the underlying cultural factors contributing to cognitive overload.

Implications for Theory and Practice

The implications of our findings extend beyond immediate organizational practices. They provide a basis for refining theoretical models of cognitive load and occupational stress. Future research can build on our results to develop more comprehensive models that account for the complex interplay of multiple factors. For practitioners, our study offers actionable insights for designing work environments and policies that safeguard cognitive resources. By balancing demands with adequate resources and implementing holistic support strategies, organizations can enhance both the quality of software production and the well-being of their developers.

In conclusion, the extensive interplay among various factors that contribute to cognitive strain necessitates a multi-faceted intervention approach. By leveraging the insights of cognitive load theory and the Job Demands-Resources model, organizations can create work practices that not only boost productivity and code quality but also promote sustainable developer well-being in today's fast-paced software development landscape.

7.1 Conclusion

The primary objective of this research was to understand the factors influencing cognitive strain during the coding phase of the Software Development Life Cycle (SDLC) and to develop a predictive model to identify cognitive strain among software developers. This was guided by two key research questions:

- **RQ1:** *What are the key factors influencing cognitive strain during the coding phase of the SDLC, and how can they be quantified for effective analysis?*
- **RQ2:** *How can the identification and management of cognitive strain improve productivity and developer well-being during the coding phase of the SDLC?*

Through a comprehensive data collection process involving software developers, we identified critical factors contributing to cognitive strain. These factors included:

- **Task Complexity:** High complexity tasks were found to significantly increase cognitive strain.
- **Work Hours:** Extended work hours correlated with higher levels of cognitive strain.
- **Task Duration:** Longer task durations were associated with increased cognitive load.
- **Stress Level:** Elevated stress levels were a strong predictor of cognitive strain.
- **Sleep Quality:** Poor sleep quality exacerbated mental fatigue and cognitive strain.
- **Multitasking:** Frequent multitasking led to divided attention and increased cognitive load.
- **Deadline Pressure:** High deadline pressure heightened stress and cognitive strain.

By designing a structured questionnaire, we quantified these subjective experiences, converting them into measurable data suitable for analysis. This addressed **RQ1** by successfully identifying and quantifying the key factors influencing cognitive strain.

The analysis revealed significant correlations between cognitive strain and variables such as high task complexity, extended work hours, poor sleep quality, frequent multitasking, and high deadline pressure.

Utilizing the collected data, we developed a predictive model using the Random Forest algorithm. The Random Forest classifier was chosen due to its ability to handle nonlinear relationships, manage interactions between variables, and its robustness against overfitting in high-dimensional data. The model achieved an accuracy of **99.1%** in predicting cognitive strain, indicating a high level of effectiveness. Important features identified by the model included:

- **Task Complexity:** The most significant predictor of cognitive strain.
- **Stress Level:** Directly correlated with cognitive strain.
- **Work Hours:** Longer hours increased the likelihood of strain.
- **Multitasking:** Frequent multitasking heightened cognitive load.

The effectiveness of the Random Forest model demonstrates its potential as a valuable tool for early identification of cognitive strain in software developers.

In addressing **RQ2**, the identification of cognitive strain through our predictive model allowed us to explore strategies and interventions that could improve productivity and developer well-being. Understanding the key factors contributing to cognitive strain, we proposed data-driven strategies such as:

- **Workload Balancing:** Adjusting workloads to prevent overworking and ensure manageable tasks.
- **Task Allocation According to Expertise:** Assigning tasks based on developers' skills and experience.
- **Flexible Scheduling:** Implementing flexible work hours to accommodate individual needs and promote adequate rest.
- **Minimizing Multitasking:** Encouraging single-task focus by minimizing interruptions and clearly prioritizing tasks.
- **Realistic Deadline Setting:** Involving developers in setting achievable timelines and including buffer periods.
- **Providing Stress Management Resources:** Offering access to stress management programs and promoting a supportive culture.

These strategies are strongly justified by both our findings and existing literature. For example, balancing workloads addresses the significant correlation between extended work hours and cognitive strain. Assigning tasks based on expertise mitigates the impact of high task complexity. Flexible scheduling improves sleep quality, which is linked to reduced cognitive strain. By proposing these interventions, we offer actionable insights that organizations can consider to enhance productivity and well-being during the coding phase of the SDLC.

The implications of this research are significant:

- **For Organizations:** The predictive model serves as a valuable tool for identifying developers at risk of cognitive strain. Implementing the proposed strategies could lead to improved productivity, higher quality outputs, and enhanced developer satisfaction.
- **For Developers:** Understanding the factors that contribute to cognitive strain empowers developers to manage their workloads and seek support proactively.
- **For the Field of Software Development:** This research contributes to a deeper understanding of cognitive strain, highlighting the importance of mental well-being in achieving optimal performance.

However, several limitations of the study must be acknowledged:

- **Sample Size and Diversity:** The study was conducted with a limited number of participants from specific organizations, which may affect the generalizability of the findings. A larger and more diverse sample would enhance the robustness of the conclusions.
- **Self-Reported Data:** The reliance on self-reported measures may introduce bias, as participants might underreport or overreport certain behaviors or experiences due to social desirability or recall inaccuracies.
- **Cross-Sectional Design:** The study's cross-sectional nature captures a snapshot in time, limiting the ability to infer causality between variables. Longitudinal studies are needed to observe changes over time and establish causal relationships.
- **External Factors:** Factors such as organizational culture, team dynamics, and personal life stressors were not included in the analysis but may significantly influence cognitive strain.

7.2 Future Work

Building on the findings and limitations of this study, several avenues for future research are proposed:

- **Expansion of the Dataset:** Collect data from a larger and more diverse population of software developers across different organizations, industries, and cultural contexts to improve the generalizability of the results.
- **Longitudinal Studies:** Conduct longitudinal research to examine how cognitive strain and its influencing factors evolve over time, allowing for the assessment of causality and long-term effects.
- **Incorporation of Additional Variables:** Include external factors such as organizational culture, management styles, and personal life stressors to provide a more comprehensive understanding of cognitive strain.

- **Testing the Proposed Interventions:** Implement and empirically test the effectiveness of the proposed strategies and interventions in real-world settings to validate their impact on productivity and well-being.
- **Integration with Real-Time Monitoring:** Explore the use of real-time data collection methods, such as physiological sensors or software usage analytics, to enhance the predictive capabilities of the model.
- **Comparative Analysis of Machine Learning Models:** Evaluate the performance of different machine learning algorithms beyond Random Forest to identify the most effective model for predicting cognitive strain.
- **Personalization of Interventions:** Investigate how personalized interventions based on individual profiles can more effectively mitigate cognitive strain.

By pursuing these future research directions, the understanding of cognitive strain in software development can be further enhanced, leading to more effective strategies for managing it.

7.3 Final Remarks

This research underscores the critical role of cognitive strain in the productivity and well-being of software developers during the coding phase of the SDLC. By identifying key factors influencing cognitive strain and developing an effective predictive model using Random Forest, we have laid the groundwork for organizations to proactively address cognitive strain.

The proposed strategies, while not experimentally tested within this study, offer a roadmap for organizations to consider in enhancing their work environments. The strong alignment between our findings and existing literature provides confidence in the potential effectiveness of these interventions.

Ultimately, addressing cognitive strain is not only beneficial for individual developers but also advantageous for organizations seeking to optimize performance and maintain a healthy, sustainable workforce. Continued research and practical application of these insights will contribute to the advancement of the software development field and the well-being of those who drive it.

References

- [1] D. Helgesson, D. Appelquist, and P. Runeson, “A grounded theory of cognitive load drivers in novice agile software development teams,” *CoRR*, vol. abs/2107.04254, 2021. [Online]. Available: <https://arxiv.org/abs/2107.04254>
- [2] D. Helgesson, E. Engström, P. Runeson, and E. Bjarnason, “Cognitive load drivers in large scale software development,” in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2019, pp. 91–94.
- [3] F. Fagerholm, M. Felderer, D. Fucci, M. Unterkalmsteiner, B. Marculescu, M. Martini, L. G. W. Tengberg, R. Feldt, B. Lehtelä, B. Nagyvárad, and J. Khattak, “Cognition in software engineering: A taxonomy and survey of a half-century of research,” *ACM Comput. Surv.*, vol. 54, no. 11s, Sep. 2022. [Online]. Available: <https://doi.org/10.1145/3508359>
- [4] M. L. Drury-Grogan, “The Changes in Team Cognition and Cognitive Artifact Use During Agile Software Development Project Management,” *Project Management Journal*, vol. 52, no. 2, pp. 127–145, Apr. 2021. [Online]. Available: <http://journals.sagepub.com/doi/10.1177/8756972820960301>
- [5] T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, and M. Züger, “Using psychophysiological measures to assess task difficulty in software development,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 402–413. [Online]. Available: <https://doi.org/10.1145/2568225.2568266>
- [6] M. Zimmer, A. Al-Yacoub, P. Ferreira, E.-M. Hubbard, and N. Lohse, “Mental Workload of Local vs Remote Operator in Human-Machine Interaction Case Study,” in *Advances in Transdisciplinary Engineering*, M. Shafik and K. Case, Eds. IOS Press, Aug. 2021. [Online]. Available: <https://ebooks.iospress.nl/doi/10.3233/ATDE210008>
- [7] A. Cao, K. K. Chintamani, A. K. Pandya, and R. D. Ellis, “NASA TLX: Software for assessing subjective mental workload,” *Behavior Research Methods*, vol. 41, no. 1, pp. 113–117, Feb. 2009. [Online]. Available: <https://doi.org/10.3758/BRM.41.1.113>
- [8] P. Leloudas, “Software Development Life Cycle,” in *Introduction to Software Testing: A Practical Guide to Testing, Design, Automation, and Execution*, P. Leloudas, Ed. Berkeley, CA: Apress, 2023, pp. 35–55. [Online]. Available: https://doi.org/10.1007/978-1-4842-9514-4_3

- [9] D. Helgesson, E. Engström, P. Runeson, and E. Bjarnason, “Cognitive Load Drivers in Large Scale Software Development,” in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, May 2019, pp. 91–94, iSSN: 2574-1837. [Online]. Available: <https://ieeexplore.ieee.org/document/8816966>
- [10] J. E. T. Akinsola, A. S. Ogunbanwo, O. J. Okesola, I. J. Odun-Ayo, F. D. Ayegbusi, and A. A. Adebisi, “Comparative Analysis of Software Development Life Cycle Models (SDLC),” in *Intelligent Algorithms in Software Engineering*, R. Silhavy, Ed. Cham: Springer International Publishing, 2020, pp. 310–322.
- [11] K. R. Lowell, “Agile Principle 2: “Welcome Changing Requirements, Even Late in Development. Agile Processes Harness Change for the Customer’s Competitive Advantage”,” in *Leading Modern Technology Teams in Complex Times: Applying the Principles of the Agile Manifesto*, K. R. Lowell, Ed. Cham: Springer International Publishing, 2023, pp. 59–65. [Online]. Available: https://doi.org/10.1007/978-3-031-36429-7_9
- [12] M. Senapathi, J. Buchan, and H. Osman, “DevOps Capabilities, Practices, and Challenges: Insights from a Case Study,” in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, Jun. 2018, pp. 57–67, arXiv:1907.10201 [cs]. [Online]. Available: <http://arxiv.org/abs/1907.10201>
- [13] O. Oni and E. Letier, “Optimizing the Incremental Delivery of Software Features Under Uncertainty,” in *Requirements Engineering: Foundation for Software Quality*, M. Daneva and O. Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 36–41.
- [14] O. E. Olorunshola and F. N. Ogwueleka, “Review of System Development Life Cycle (SDLC) Models for Effective Application Delivery,” in *Information and Communication Technology for Competitive Strategies (ICTCS 2020)*, A. Joshi, M. Mahmud, R. G. Ragel, and N. V. Thakur, Eds. Singapore: Springer, 2022, pp. 281–289.
- [15] L. Gonçalves and K. Farias, “Towards the measurement of mental effort in software engineering: A research agenda,” *International Journal of Computer Applications*, vol. 177, pp. 1–8, 01 2020.
- [16] L. A. DeChurch and J. Mesmer-Magnus, “The cognitive underpinnings of effective teamwork: a meta-analysis.” *Journal of Applied Psychology*, vol. 95, pp. 32–53, 2010.
- [17] D. McEwan, G. R. Ruissen, M. A. Eys, B. D. Zumbo, and M. R. Beauchamp, “The Effectiveness of Teamwork Training on Teamwork Behaviors and Team Performance: A Systematic Review and Meta-Analysis of Controlled Interventions,” *PLOS ONE*, vol. 12, no. 1, p. e0169604, Jan. 2017. [Online]. Available: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0169604>
- [18] L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: <http://dx.doi.org/10.1023/A%3A1010933404324>

- [19] A. Liaw and M. Wiener, “Classification and Regression by randomForest,” *R News*, vol. 2, no. 3, pp. 18–22, 2002. [Online]. Available: <http://CRAN.R-project.org/doc/Rnews/>
- [20] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, “Bias in random forest variable importance measures: Illustrations, sources and a solution.” *bmc bioinformatics*, 8(1), 25,” *BMC bioinformatics*, vol. 8, p. 25, 02 2007.
- [21] H. Lin, J. Jia, J. Qiu, Y. Zhang, G. Shen, L. Xie, J. Tang, L. Feng, and T.-S. Chua, “Detecting stress based on social interactions in social networks,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 9, pp. 1820–1833, 2017.
- [22] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, pp. 1735–80, 12 1997.
- [23] F. A. Gers, J. A. Schmidhuber, and F. A. Cummins, “Learning to forget: Continual prediction with lstm,” *Neural Comput.*, vol. 12, no. 10, p. 2451–2471, Oct. 2000. [Online]. Available: <https://doi.org/10.1162/089976600300015015>
- [24] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, vol. 38, 03 2013.
- [25] D. Das Chakladar, S. Dey, P. Roy, and D. Dogra, “Eeg-based mental workload estimation using deep blstm-lstm network and evolutionary algorithm,” *Biomedical Signal Processing and Control*, vol. 60, p. 101989, 07 2020.
- [26] D. Hosmer, S. Lemeshow, and R. Sturdivant, *Applied Logistic Regression*, ser. Wiley Series in Probability and Statistics. Wiley, 2013. [Online]. Available: <https://books.google.se/books?id=64JYAwAAQBAJ>
- [27] S. Menard, *Applied Logistic Regression Analysis*, ser. Applied Logistic Regression Analysis. SAGE Publications, 2002. [Online]. Available: <https://books.google.se/books?id=EAI1QmUUsbUC>
- [28] A. Agresti, *An Introduction to Categorical Data Analysis*, ser. Wiley Series in Probability and Statistics. Wiley, 2018. [Online]. Available: <https://books.google.se/books?id=ukNxDwAAQBAJ>
- [29] Z. Izdebski, A. Kozakiewicz, M. Białorudzki, J. Dec-Pietrowska, and J. Mazur, “Occupational burnout in healthcare workers, stress and other symptoms of work overload during the covid-19 pandemic in poland,” *International Journal of Environmental Research and Public Health*, vol. 20, no. 3, 2023. [Online]. Available: <https://www.mdpi.com/1660-4601/20/3/2428>
- [30] N. Altman, “An introduction to kernel and nearest-neighbor nonparametric regression,” *American Statistician*, vol. 46, no. 3, pp. 175–185, Aug. 1992, funding Information: *N. S. Altman is Assistant Professor, Biometrics Unit, Cornell University, Ithaca, NY 14853. Preparation of this article was partially funded by Hatch Grant 151410 NYF. The author thanks C. McCulloch for comments that substantially improved this article and L. Molinari and H. Henderson for providing the data used in Examples B and C. Two anonymous referees and

an associate editor provided numerous comments that substantially improved the presentation of the material.

- [31] T. Cover and P. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.
- [32] L. E. Peterson, “K-nearest neighbor,” *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009, revision #137311.
- [33] R. Duda, P. Hart, and D. Stork, *Pattern Classification*. Wiley, 2012. [Online]. Available: <https://books.google.se/books?id=Br33IRC3PkQC>
- [34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008. [Online]. Available: <http://arxiv.org/abs/1706.03762>
- [35] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, p. 273–297, Sep. 1995. [Online]. Available: <https://doi.org/10.1023/A:1022627411411>
- [36] I. Rish, “An empirical study of the naïve bayes classifier,” *IJCAI 2001 Work Empir Methods Artif Intell*, vol. 3, 01 2001.
- [37] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 785–794. [Online]. Available: <https://doi.org/10.1145/2939672.2939785>
- [38] D. Alawad, M. Panta, M. Zibran, and M. R. Islam, “An Empirical Study of the Relationships between Code Readability and Software Complexity,” Aug. 2019, arXiv:1909.01760 [cs]. [Online]. Available: <http://arxiv.org/abs/1909.01760>
- [39] J. Huang and C. Zhang, “Debugging Concurrent Software: Advances and Challenges,” *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 861–868, Sep. 2016. [Online]. Available: <https://doi.org/10.1007/s11390-016-1669-8>
- [40] M. Czerwinski, E. Horvitz, and S. Wilhite, “A diary study of task switching and interruptions,” *Conference on Human Factors in Computing Systems - Proceedings*, vol. 6, 02 2004.
- [41] M. M. Ajmal, M. Khan, A. Gunasekaran, and P. Helo, “Managing project scope creep in construction industry,” *Engineering, Construction and Architectural Management*, vol. 29, pp. 2786–2809, 2021.
- [42] M. C. Hu, N. Muhlert, N. Robertson, and M. Winter, “Fatigue and cognitive performance change in ms: multifactorial with disparate influences short title: predictors of fatigue and performance change in ms,” 2018.
- [43] D. Bläsing and M. Bornewasser, “Influence of Increasing Task Complexity and Use of Informational Assistance Systems on Mental Workload,” *Brain Sciences*, vol. 11, no. 1, p. 102, Jan. 2021. [Online]. Available: <https://www.mdpi.com/2076-3425/11/1/102>

- [44] V. Kalakoski, S. Selinheimo, T. Valtonen, J. Turunen, S. Käpykangas, H. Ylisassi, P. Toivio, H. Järnefelt, H. Hannonen, and T. Paaajanen, “Effects of a cognitive ergonomics workplace intervention (CogErg) on cognitive strain and well-being: a cluster-randomized controlled trial. A study protocol,” *BMC Psychology*, vol. 8, no. 1, p. 1, Jan. 2020. [Online]. Available: <https://doi.org/10.1186/s40359-019-0349-1>
- [45] “An Investigation into Bad Smells in Model-Based Systems Engineering - ProQuest.” [Online]. Available: <https://www.proquest.com/openview/197642c9490f064f805ad1e074225c95/gscholarcbl=18750diss=y>
- [46] A. Alzayed and A. Khalfan, “Understanding Top Management Involvement in SDLC Phases,” *Journal of Software*, pp. 87–120, May 2022. [Online]. Available: <http://www.jsoftware.us/index.php?m=content&c=index&a=show&catid=235&id=3071>
- [47] L. Gonçalves, K. Farias, B. da Silva, and J. Fessler, “Measuring the cognitive load of software developers: A systematic mapping study,” 03 2019.
- [48] B. Abu-Shaqra, “Technoethics and Sensemaking: Risk Assessment and Knowledge Management of Ethical Hacking in a Sociotechnical Society,” Ph.D. dissertation, Université d’Ottawa / University of Ottawa, Apr. 2020. [Online]. Available: <http://hdl.handle.net/10393/40393>
- [49] L.-M. Bogza, C. Patry-Lebeau, E. Farmanova, H. O. Witteman, J. Elliott, P. Stolee, C. Hudon, and A. M. C. Giguere, “User-Centered Design and Evaluation of a Web-Based Decision Aid for Older Adults Living With Mild Cognitive Impairment and Their Health Care Providers: Mixed Methods Study,” *Journal of Medical Internet Research*, vol. 22, no. 8, p. e17406, Aug. 2020. [Online]. Available: <https://www.jmir.org/2020/8/e17406>
- [50] F. Xu and L. Jin, “Impact of daily entrepreneurial stressors on long-term transformational leader behaviors and well-being: Differences in experienced and nascent entrepreneurs,” *Journal of Business Research*, vol. 139, pp. 280–291, Feb. 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0148296321007116>
- [51] J. A. Granek, W. Merchant, N. Pavlovic, C. Apostoli, O. Vartanian, S. de Youngster, and M. Peart, “Usage analytics and end-user feedback for the road to mental readiness (r2mr) mobile application,” 2022.
- [52] G. Dlamini, S. Ergasheva, Z. Kholmatova, A. Kruglov, A. Sadovykh, G. Succi, A. Timchenko, X. Vasquez, and E. Zouev, “Metrics for Software Process Quality Assessment in the Late Phases of SDLC,” in *Intelligent Computing*, K. Arai, Ed. Cham: Springer International Publishing, 2022, pp. 639–655.
- [53] A. K. Kadam, K. H. Krishna, N. Varshney, A. Deepak, H. S. Pokhariya, S. K. Hegde, and V. H. Patil, “Design of Software Reliability Growth Model for Improving Accuracy in the Software Development Life Cycle (SDLC),” *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 1s, pp. 38–50, 2024. [Online]. Available: <https://ijisae.org/index.php/IJISAE/article/view/3393>

- [54] I. Härkönen, “Wearable Assistance System for Early Phase of Post-stroke Rehabilitation,” Mar. 2020. [Online]. Available: <https://aaltodoc.aalto.fi/handle/123456789/43521>
- [55] S. Najihi, S. Elhadi, R. A. Abdelouahid, and A. Marzak, “Software Testing from an Agile and Traditional view,” *Procedia Computer Science*, vol. 203, pp. 775–782, Jan. 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050922007219>
- [56] B. Sugiantoro, M. Anshari, and D. Sudrajat, “Developing Framework for Web Based e-Commerce: Secure-SDLC,” *Journal of Physics: Conference Series*, vol. 1566, no. 1, p. 012020, Jun. 2020. [Online]. Available: <https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012020>
- [57] D. Hoti, M. Maloku, and K. Gashi, *SDLC Phases of a Mobile Application*, 06 2023, pp. 232–249.
- [58] J. M. Carey and C. J. Kacmar, “The impact of communication mode and task complexity on small group performance and member satisfaction,” *Computers in Human Behavior*, vol. 13, no. 1, pp. 23–49, 1997. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0747563296000271>
- [59] J. A. Espinosa, S. A. Slaughter, R. E. Kraut, and J. D. Herbsleb, “Familiarity, Complexity, and Team Performance in Geographically Distributed Software Development,” *Organization Science*, vol. 18, no. 4, pp. 613–630, August 2007. [Online]. Available: <https://ideas.repec.org/a/inm/ororsc/v18y2007i4p613-630.html>
- [60] B. M. Knisely, J. S. Joyner, and M. Vaughn-Cooke, “Cognitive task analysis and workload classification,” *MethodsX*, vol. 8, p. 101235, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2215016121000285>
- [61] V. Leso, L. Fontana, A. Caturano, I. Vetrani, M. Fedele, and I. Iavicoli, “Impact of shift work and long working hours on worker cognitive functions: Current evidence and future research needs,” *International Journal of Environmental Research and Public Health*, vol. 18, no. 12, 2021. [Online]. Available: <https://www.mdpi.com/1660-4601/18/12/6540>
- [62] T. Wei, W. Wang, and S. Yu, “Analysis of the cognitive load of employees working from home and the construction of the telecommuting experience balance model,” *Sustainability*, vol. 14, no. 18, 2022. [Online]. Available: <https://www.mdpi.com/2071-1050/14/18/11722>
- [63] A. Rhéaume and J. Mullen, “The impact of long work hours and shift work on cognitive errors in nurses,” *Journal of Nursing Management*, vol. 26, p. 26–32, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:4921999>
- [64] J. Fan and A. P. Smith, “The impact of workload and fatigue on performance,” in *International Symposium on Human Mental Workload*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:23234290>
- [65] S. Jeuris and J. E. Bardram, “Dedicated workspaces: Faster resumption times and reduced cognitive load in sequential multitasking,” *Computers*

- in *Human Behavior*, vol. 62, pp. 404–414, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0747563216302308>
- [66] M. Virtanen, A. Singh-Manoux, J. Ferrie, D. Gimeno, M. Marmot, M. Elovainio, M. Jokela, J. Vahtera, and M. Kivimäki, “Long working hours and cognitive function the whitehall ii study,” *American journal of epidemiology*, vol. 169, pp. 596–605, 02 2009.
- [67] S. Lee, J. Y. Choi, and W. Lee, “The impact of long working hours on cognitive function: A follow-up study with gender stratification,” *Journal of Alzheimer’s Disease*, vol. 80, no. 2, pp. 727–734, 2021, pMID: 33579851. [Online]. Available: <https://doi.org/10.3233/JAD-201404>
- [68] M. L. Reyes and J. D. Lee, “Effects of cognitive load presence and duration on driver eye movements and event detection performance,” *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 11, no. 6, pp. 391–402, 2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1369847808000211>
- [69] J. A. Walker, M. Aswad, and G. Lacroix, “The impact of cognitive load on prospective and retrospective time estimates at long durations: An investigation using a visual and memory search paradigm,” *Memory & Cognition*, vol. 50, no. 4, pp. 837–851, May 2022. [Online]. Available: <https://doi.org/10.3758/s13421-021-01241-7>
- [70] E. E. Miller, L. N. Boyle, J. W. Jenness, and J. D. Lee, “Voice control tasks on cognitive workload and driving performance: Implications of modality, difficulty, and duration,” *Transportation Research Record*, vol. 2672, no. 37, pp. 84–93, 2018. [Online]. Available: <https://doi.org/10.1177/0361198118797483>
- [71] J. Li, Y. Zhou, and T. Hao, “Effects of the interaction between time-on-task and task load on response lapses,” *Behavioral Sciences*, vol. 14, no. 11, 2024. [Online]. Available: <https://www.mdpi.com/2076-328X/14/11/1086>
- [72] G. Borragán, H. Slama, M. Bartolomei, and P. Peigneux, “Cognitive fatigue: A time-based resource-sharing account,” *Cortex*, vol. 89, pp. 71–84, 2017. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0010945217300370>
- [73] D. Conway, I. Dick, Z. Li, Y. Wang, and F. Chen, “The effect of stress on cognitive load measurement,” in *Human-Computer Interaction – INTERACT 2013*, P. Kotzé, G. Marsden, G. Lindgaard, J. Wesson, and M. Winckler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 659–666.
- [74] C. L. Bong, K. Fraser, and D. Oriot, *Cognitive Load and Stress in Simulation*. Cham: Springer International Publishing, 2016, pp. 3–17. [Online]. Available: https://doi.org/10.1007/978-3-319-24187-6_1
- [75] F. Chen, J. Zhou, Y. Wang, K. Yu, S. Z. Arshad, A. Khawaji, and D. Conway, *Stress and Cognitive Load*. Cham: Springer International Publishing, 2016, pp. 185–194. [Online]. Available: https://doi.org/10.1007/978-3-319-31700-7_12
- [76] A. Niculescu, Y. Cao, and A. Nijholt, “Stress and cognitive load in multimodal conversational interactions,” 05 2010.

- [77] V. Markova, T. Ganchev, and K. Kalinkov, "Clas: A database for cognitive load, affect and stress recognition," in *2019 International Conference on Biomedical Innovations and Applications (BIA)*, 2019, pp. 1–4.
- [78] C. Sandi, "Stress and cognition," *WIREs Cognitive Science*, vol. 4, no. 3, pp. 245–261, 2013. [Online]. Available: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcs.1222>
- [79] M. Steinhauser, M. Maier, and R. Hübner, "Cognitive control under stress: How stress affects strategies of task-set reconfiguration," *Psychological Science*, vol. 18, no. 6, pp. 540–545, 2007, pMID: 17576268. [Online]. Available: <https://doi.org/10.1111/j.1467-9280.2007.01935.x>
- [80] E. Galy, M. Cariou, and C. Mélan, "What is the relationship between mental workload factors and cognitive load types?" *International Journal of Psychophysiology*, vol. 83, no. 3, pp. 269–275, 2012. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S016787601100300X>
- [81] K. L. Fraser, P. Ayres, and J. Sweller, "Cognitive load theory for the design of medical simulations," *Simulation in Healthcare*, vol. 10, no. 5, 2015. [Online]. Available: https://journals.lww.com/simulationinhealthcare/fulltext/2015/10000/cognitive_load_theory_for_the_design_of_medical.7.aspx
- [82] O. Benkirane, B. Delwiche, O. Mairesse, and P. Peigneux, "Impact of sleep fragmentation on cognition and fatigue," *International Journal of Environmental Research and Public Health*, vol. 19, no. 23, 2022. [Online]. Available: <https://www.mdpi.com/1660-4601/19/23/15485>
- [83] O. Benkirane, P. Simor, O. Mairesse, and P. Peigneux, "Sleep fragmentation modulates the neurophysiological correlates of cognitive fatigue," *Clocks amp; Sleep*, vol. 6, no. 4, pp. 602–618, 2024. [Online]. Available: <https://www.mdpi.com/2624-5175/6/4/41>
- [84] S. Miyata, A. Noda, K. Iwamoto, N. Kawano, M. Okuda, and N. Ozaki, "Poor sleep quality impairs cognitive performance in older adults," *Journal of Sleep Research*, vol. 22, no. 5, pp. 535–541, 2013. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/jsr.12054>
- [85] P. Alhola and P. Polo-Kantola, "Sleep deprivation: Impact on cognitive performance," *Neuropsychiatric Disease and Treatment*, vol. 3, no. 5, pp. 553–567, 2007, pMID: 19300585. [Online]. Available: <https://www.tandfonline.com/doi/abs/10.2147/ndt.s12160203>
- [86] P. Maquet, "The role of sleep in learning and memory," *Science*, vol. 294, no. 5544, pp. 1048–1052, 2001. [Online]. Available: <https://www.science.org/doi/abs/10.1126/science.1062856>
- [87] R. Stickgold, "Sleep-dependent memory consolidation," *Nature*, vol. 437, no. 7063, pp. 1272–1278, Oct 2005. [Online]. Available: <https://doi.org/10.1038/nature04286>
- [88] J. J. Pilcher and A. I. Huffcutt, "Effects of sleep deprivation on performance: A meta-analysis," *Sleep*, vol. 19, no. 4, pp. 318–326, 06 1996. [Online]. Available: <https://doi.org/10.1093/sleep/19.4.318>

- [89] I. Philibert, "Sleep loss and performance in residents and nonphysicians: A meta-analytic examination," *Sleep*, vol. 28, no. 11, pp. 1392–1402, 11 2005. [Online]. Available: <https://doi.org/10.1093/sleep/28.11.1392>
- [90] J. C. Lo, J. A. Groeger, G. H. Cheng, D.-J. Dijk, and M. W. Chee, "Self-reported sleep duration and cognitive performance in older adults: a systematic review and meta-analysis," *Sleep Medicine*, vol. 17, pp. 87–98, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1389945715019796>
- [91] G. Zhu, F. Zong, H. Zhang, B. Wei, and F. Liu, "Cognitive load during multitasking can be accurately assessed based on single channel electroencephalography using graph methods," *IEEE Access*, vol. 9, pp. 33 102–33 109, 2021.
- [92] Özgür Örün and Y. Akbulut, "Effect of multitasking, physical environment and electroencephalography use on cognitive load and retention," *Computers in Human Behavior*, vol. 92, pp. 216–229, 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0747563218305624>
- [93] U. Tugtekin and H. F. Odabasi, "Effect of multitasking and task characteristics interaction on cognitive load and learning outcomes in virtual reality learning environments," *Education and Information Technologies*, vol. 28, no. 11, pp. 14915–14942, Nov 2023. [Online]. Available: <https://doi.org/10.1007/s10639-023-11813-6>
- [94] J. Aagaard, "Media multitasking, attention, and distraction: a critical discussion," *Phenomenology and the Cognitive Sciences*, vol. 14, no. 4, pp. 885–896, Dec 2015. [Online]. Available: <https://doi.org/10.1007/s11097-014-9375-x>
- [95] E. Altmann and W. Gray, "An integrated model of cognitive control in task switching," *Psychological review*, vol. 115, pp. 602–39, 07 2008.
- [96] L. M. Carrier, L. D. Rosen, N. A. Cheever, and A. F. Lim, "Causes, effects, and practicalities of everyday multitasking," *Developmental Review*, vol. 35, pp. 64–78, 2015, special Issue: Living in the "Net" Generation: Multitasking, Learning, and Development. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0273229714000513>
- [97] S. Chinchachokchai, B. R. Duff, and S. Sar, "The effect of multitasking on time perception, enjoyment, and ad evaluation," *Computers in Human Behavior*, vol. 45, pp. 185–191, 2015. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0747563214007055>
- [98] M. L. Courage, A. Bakhtiar, C. Fitzpatrick, S. Kenny, and K. Brandeau, "Growing up multitasking: The costs and benefits for cognitive development," *Developmental Review*, vol. 35, pp. 5–41, 2015, special Issue: Living in the "Net" Generation: Multitasking, Learning, and Development. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0273229714000483>
- [99] M. Dindar and Y. Akbulut, "Effects of multitasking on retention and topic interest," *Learning and Instruction*, vol. 41, pp. 94–105, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0959475215300372>

- [100] M. Kuuttila, M. Mäntylä, U. Farooq, and M. Claes, “Time pressure in software engineering: A systematic review,” *Information and Software Technology*, vol. 121, p. 106257, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0950584920300045>
- [101] R. Gilal, M. Omar, and M. Md Rejab, “The key factors contribute to time pressure in software development projects: A review,” *International Journal of ADVANCED AND APPLIED SCIENCES*, vol. 10, pp. 155–165, 10 2023.
- [102] M. V. Mäntylä, K. Petersen, T. O. A. Lehtinen, and C. Lassenius, “Time pressure: a controlled experiment of test case development and requirements review,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 83–94. [Online]. Available: <https://doi.org/10.1145/2568225.2568245>
- [103] B. Penzenstadler, “Leverage points for focus flow and communitas,” in *Proceedings of the 7th International Conference on ICT for Sustainability*, ser. ICT4S2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 267–274. [Online]. Available: <https://doi.org/10.1145/3401335.3401826>
- [104] M. Kuuttila, “Time pressure and well-being in software engineering: evidence from software repositories, experience sampling, and prior literature,” 2021.
- [105] G. Williams and A. Mahmoud, “Analyzing, classifying, and interpreting emotions in software users’ tweets,” in *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, 2017, pp. 2–7.
- [106] M. Kuuttila, M. V. Mäntylä, M. Claes, and M. Elovainio, “Daily questionnaire to assess self-reported well-being during a software development project,” in *Proceedings of the 3rd International Workshop on Emotion Awareness in Software Engineering*, ser. SEmotion ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 39–43. [Online]. Available: <https://doi.org/10.1145/3194932.3194942>
- [107] M. Kuuttila, M. V. Mantyla, U. Farooq, and M. Claes, “What do we know about time pressure in software development?” *IEEE Software*, vol. 38, no. 5, pp. 32–38, 2021.
- [108] N. Koundal, A. Abdalhadi, M. S. Al-Quraishi, I. Elamvazuthi, M. S. Moosavi, C. Guillet, F. Merienne, and N. M. Saad, “Effect of interruptions and cognitive demand on mental workload: A critical review,” *IEEE Access*, vol. 12, pp. 54 405–54 425, 2024.
- [109] M. Wu, Q. Gao, and Y. Liu, “Exploring the effects of interruptions in different phases of complex decision-making tasks,” *Human Factors*, vol. 65, no. 3, pp. 450–481, 2023, PMID: 34061699. [Online]. Available: <https://doi.org/10.1177/00187208211018882>
- [110] M. Wirzberger, J. P. Borst, J. F. Krems, and G. D. Rey, “Memory-related cognitive load effects in an interrupted learning task: A model-based

- explanation,” *Trends in Neuroscience and Education*, vol. 20, p. 100139, 2020. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2211949320300156>
- [111] J. Lindblom and J. Gündert, “Managing mediated interruptions in manufacturing: Selected strategies used for coping with cognitive load,” in *Advances in Neuroergonomics and Cognitive Engineering*, K. S. Hale and K. M. Stanney, Eds. Cham: Springer International Publishing, 2017, pp. 389–403.
- [112] F. Schaule, J. O. Johanssen, B. Bruegge, and V. Loftness, “Employing consumer wearables to detect office workers’ cognitive load for interruption management,” *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, no. 1, Mar. 2018. [Online]. Available: <https://doi.org/10.1145/3191764>
- [113] J. Sherbino and G. Norman, “Task switching, multitasking, and errors: A psychologic perspective on the impact of interruptions,” *Annals of Emergency Medicine*, vol. 78, pp. 425–428, 09 2021.
- [114] L. Becker, H. C. Kaltenecker, D. Nowak, M. Weigl, and N. Rohleder, “Biological stress responses to multitasking and work interruptions: A randomized controlled trial,” *Psychoneuroendocrinology*, vol. 156, p. 106358, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0306453023003360>
- [115] X. Yuan and L. Zhong, “Effects of multitasking and task interruptions on task performance and cognitive load: considering the moderating role of individual resilience,” *Current Psychology*, vol. 43, pp. 1–11, 05 2024.
- [116] J. Medeiros, R. Couceiro, G. Duarte, J. Durães, J. Castelhana, C. Duarte, M. Castelo-Branco, H. Madeira, P. de Carvalho, and C. Teixeira, “Can eeg be adopted as a neuroscience reference for assessing software programmers’ cognitive load?” *Sensors*, vol. 21, no. 7, 2021. [Online]. Available: <https://www.mdpi.com/1424-8220/21/7/2338>
- [117] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli, “Information needs in contemporary code review,” *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, Nov. 2018. [Online]. Available: <https://doi.org/10.1145/3274404>
- [118] P. W. Gonçalves, E. Fregnan, T. Baum, K. Schneider, and A. Bacchelli, “Do explicit review strategies improve code review performance?” in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 606–610. [Online]. Available: <https://doi.org/10.1145/3379597.3387509>
- [119] N. J. Cooke, J. C. Gorman, C. W. Myers, and J. L. Duran, “Interactive team cognition,” *Cognitive Science*, vol. 37, no. 2, pp. 255–285, 2013. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12009>
- [120] M. A. Khawaja, F. Chen, and N. Marcus, “Analysis of collaborative communication for linguistic cues of cognitive load,” *Human Factors*, vol. 54, no. 4, pp. 518–529, 2012, pMID: 22908676. [Online]. Available: <https://doi.org/10.1177/0018720811431258>

- [121] M. M. Willett and M. Demir, “Understanding the impact of team cognitive load and advice compliance in urban search and rescue task,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 67, no. 1, pp. 2484–2489, 2023. [Online]. Available: <https://doi.org/10.1177/21695067231192293>
- [122] P. Liu, A. Lyndon, J. L. Holl, J. Johnson, K. Y. Bilimoria, and A. M. Stey, “Barriers and facilitators to interdisciplinary communication during consultations: a qualitative study,” *BMJ Open*, vol. 11, no. 9, 2021. [Online]. Available: <https://bmjopen.bmj.com/content/11/9/e046111>
- [123] M. J. Scalia, S. Zhou, D. A. Grimm, J. L. Harrison, and J. C. Gorman, “The role of timing of information front-loading and planning ahead in all-human vs. human-autonomy team performance,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 66, no. 1, pp. 530–534, 2022. [Online]. Available: <https://doi.org/10.1177/1071181322661251>
- [124] R. McKendrick, T. Shaw, H. Saqer, E. de Visser, and R. Parasuraman, “Team performance and communication within networked supervisory control human-machine systems,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 55, no. 1, pp. 262–266, 2011. [Online]. Available: <https://doi.org/10.1177/1071181311551054>
- [125] Y. Wu, Z. Zhang, Y. Zhang, B. Zheng, and F. Aghazadeh, “Pupil response in visual tracking tasks: The impacts of task load, familiarity, and gaze position,” *Sensors*, vol. 24, no. 8, 2024. [Online]. Available: <https://www.mdpi.com/1424-8220/24/8/2545>
- [126] J. Sweller, J. J. G. Van Merriënboer, and F. Paas, “Cognitive architecture and instructional design: 20 years later,” *Educational Psychology Review*, vol. 31, pp. 261–292, 06 2019.
- [127] B. Dillon and R. Thompson, “Software development and tool usability,” in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 2016, pp. 1–4.
- [128] R. Karasek and T. Theorell, *Healthy Work*. Basic Books, 1990. [Online]. Available: <https://books.google.se/books?id=MrXuAAAAMAAJ>
- [129] E. Demerouti, F. Nachreiner, and W. Schaufeli, “The job demands–resources model of burnout,” *The Journal of applied psychology*, vol. 86, pp. 499–512, 06 2001.
- [130] H. Xi, D. Xu, and Y. Son, “Dynamic scheduling and workforce assignment in open source software development,” *CoRR*, vol. abs/2009.09168, 2020. [Online]. Available: <https://arxiv.org/abs/2009.09168>
- [131] J. Edwards, *Person-job fit: A conceptual integration, literature review, and methodological critique.*, 1991st ed., 1991, vol. 6, pp. 283–357.
- [132] J. Sweller, “Cognitive load during problem solving: Effects on learning,” *Cognitive Science*, vol. 12, no. 2, pp. 257–285, 1988. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/0364021388900237>

- [133] S. Folkard and T. Monk, *Hours of Work: Temporal Factors in Work Scheduling*, ser. Wiley series in psychology and productivity at work. Wiley, 1985. [Online]. Available: <https://books.google.se/books?id=L-LHPQAACAAJ>
- [134] E. Hill, J. Grzywacz, S. Allen, V. Blanchard, C. Matz, S. Shulkin, and M. Pitt-Catsouphes, “Defining and conceptualizing workplace flexibility,” *Community*, vol. Work and Family, pp. 149–163, 05 2008.
- [135] J. Rubinstein, D. Meyer, and J. Evans, “Executive control of cognitive processes in task switching,” *Journal of experimental psychology. Human perception and performance*, vol. 27, pp. 763–97, 08 2001.
- [136] S. Leroy and A. M. Schmidt, “The effect of regulatory focus on attention residue and performance during interruptions,” *Organizational Behavior and Human Decision Processes*, vol. 137, pp. 218–235, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0749597816304630>
- [137] N. Butt and N. Warraich, “Multitasking behavior in the workplace: A systematic review,” *Journal of Social Research Development*, vol. 3, pp. 229–247, 12 2022.
- [138] E. Locke and G. Latham, “Building a practically useful theory of goal setting and task motivation: A 35year odyssey,” *American Psychologist - AMER PSYCHOL*, vol. 57, pp. 705–717, 09 2002.
- [139] R. Lazarus and S. Folkman, *Stress, Appraisal, and Coping*. Springer Publishing Company, 1984. [Online]. Available: <https://books.google.se/books?id=i-ySQQuUpr8C>
- [140] D. Basten, M. Müller, M. Ott, O. Pankratz, and C. Rosenkranz, “Impact of time pressure on software quality: A laboratory experiment on a game-theoretical model,” *PLoS One*, vol. 16, no. 1, p. e0245599, Jan. 2021.
- [141] R. D. Austin, “The effects of time pressure on quality in software development: An agency model,” *Information Systems Research*, vol. 12, no. 2, pp. 195–207, 2001. [Online]. Available: <https://doi.org/10.1287/isre.12.2.195.9699>
- [142] K. Richardson and H. Rothstein, “Effects of occupational stress management intervention programs: A meta-analysis,” *Journal of occupational health psychology*, vol. 13, pp. 69–93, 01 2008.
- [143] R. Eisenberger, F. Stinglhamber, C. Vandenberghe, I. Sucharski, and L. Rhoades, “Perceived supervisor support: Contributions to perceived organizational support and employee retention,” *The Journal of applied psychology*, vol. 87, pp. 565–73, 07 2002.
- [144] Pfeiffer, Sauer, and Ritter, “Agile methods as stress management tools? an empirical study,” *Work Organisation, Labour Globalisation*, vol. 13, p. 20, 01 2019.
- [145] J. Suárez and A. Vizcaíno, “Stress, motivation, and performance in global software engineering,” *Journal of Software: Evolution and Process*, vol. 36, no. 5, p. e2600, 2024. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2600>

- [146] E. Demerouti, "The job demands-resources model: State of the art," *Journal of Managerial Psychology*, vol. 22, pp. 309–328, 04 2007.
- [147] H. Selye, *Stress without Distress*. Boston, MA: Springer US, 1976, pp. 137–146. [Online]. Available: https://doi.org/10.1007/978-1-4684-2238-2_9

Appendix A

Supplemental Information

The relevant codes and results can be found in the given link:

Pericherla, S. B. R., Porandla, M. (2025). *Cognitive_strain[Dataset].Zenodo*.<https://doi.org/10.5281/zenodo.14841546>

