Bachelor's Thesis

Input Size: A Crucial Factor for Program-Comprehension Experiments?

Cara Blasius

November 29, 2024

Advisors:

Norman Peitek	Chair of Software Engineering
Anna-Maria Maurer	Chair of Software Engineering

Examiners:

Prof. Dr. Sven Apel	Chair of Software Engineering
Prof. Dr. Vera Demberg	Chair of Computer Science and Computational Linguistics

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Research on program comprehension has increased drastically over the years, and more and more studies are choosing recursive code as a subject of interest. Arbitrary input sizes endanger the validity of those experiments since the effect of specific input sizes on performance has not been explored yet. A systematic investigation is needed to analyze the impact of input size on program comprehension. Our experiment fills that gap by providing insight into how and to what extent the input size of a recursive function affects the correctness, response time, and eye movements of participants. We conducted a controlled laboratory experiment involving twelve computer science undergraduates with comparable programming experience. Each participant was asked to calculate the output of 15 recursive Java code snippets, which varied in recursion type and input size. During the experiment, we measured correctness, response time, and visual attention. The results revealed significant effects correlating with cognitive load. Our analysis indicates that larger input sizes can significantly reduce correctness and increase response times. Furthermore, tree recursive functions were shown to strengthen these effects, especially at higher input sizes. Apart from the number of fixations, we detected no significant differences in visual attention. Our results can serve as a basis for determining suitable input sizes for recursive functions in upcoming program comprehension experiments, potentially enhancing their validity, methodology, and comparability.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Program comprehension is a crucial cognitive process that involves understanding and interpreting code. Today, there is more research on the topic than ever. The number of papers published on the subject increased drastically over the last few years [48]. By understanding the cognitive process, researchers aim to provide suitable tools, programming languages, and coding conventions to support developers in their everyday work [37]. Moreover, understanding the cognitive process of program comprehension is essential to improve the teaching of programming concepts [37].
One key programming concept is recursion. A recursive function is specified as a function that calls itself in its body [32]. At many universities, it is comprehensively taught early on in the first few computer science lectures and is widely spread among advanced software development [32]. Due to its importance, several software engineering studies use recursion with specific inputs as a material in their code snippet selection [1, 20, 28]. However, none of them consider the effect of different input sizes on their results since this has not been systematically analyzed in research yet.

To fill this research gap and gather information about the subject, we conducted an experiment to evaluate the effect of different input sizes of recursive functions on program comprehension. We conducted a controlled lab experiment involving twelve computer science undergraduates with similar programming experience. Specifically, we consider correctness, response time, and visual attention during the program comprehension process to understand *how* and *to what extent* different input size affects those variables. In former times, software engineering research was forced to rely on surveys and think-aloud protocols to measure task performance [35]. Today, in the neuroage, a variety of measuring tools are available, among which eye-tracking stands out as a valuable resource [3, 27]. Today, eye-tracking has been widely recognized as a standard tool for the examination of program comprehension [36]. By utilizing eye-tracking technology, we can effectively monitor visual attention and analyze cognitive effort through the collection of eye movement data [30]. Measuring correctness, response time, and visual attention enables us to assess the cognitive demands novice programmers face when understanding recursive functions with various input sizes.

Our results indicate that larger input sizes in recursive functions significantly increase response times and the number of fixations and reduce correctness among novice programmers. Although eye-tracking data showed trends of longer fixation durations and a rising number of fixations per second with larger inputs, these changes were not statistically significant. Furthermore, we found that the type of recursion affects the effect of input size on the correctness of novice programmers. Especially for tasks with tree recursive

functions, correctness declined sharply as input size grew compared to other types of recursive functions.

Multiple studies use an unspecific input n for their recursive code snippets [23, 46]. With research about specific inputs, we aim to improve the comparability of future program comprehension studies using recursive code by supporting the determination of a suitable input size for their applied recursive functions. While there is an increase in papers discussing threats to validity in program comprehension regarding threats [34] such as identifier naming, experience, and eye tracking [48], none of those mention the specific choice of input size. By gathering more information on the topic, our experiment allows for better isolation of results, which enhances the validity and methodology of future program comprehension studies. In addition to improving general experiments on program comprehension, our research offers valuable insights for future investigations into the impact of input size on program comprehension. A more in-depth analysis could benefit from the inclusion of additional eye-tracking metrics, such as linearity and reading order, to gain a more comprehensive understanding of visual attention patterns. Furthermore, expanding the participant pool beyond the scope of this study would enhance the statistical power of results, enabling the detection of more subtle effects that may have been overlooked in our current investigation. These extensions will contribute to a deeper and more nuanced understanding of how input size and type of recursion collectively influence program comprehension among novice programmers.

2 Background and Related Work

This chapter outlines fundamental concepts that are essential for understanding the relationship between input size and program comprehension in novice programmers. It addresses types of recursion and the cognitive load theory and gives an introduction to the fundamentals of eye-tracking.

2.1 Recursion

Recursion is an important mathematical concept widely spread in software development [31]. A recursive function is specified as a function that calls itself in their body [14].

2.1.1 Types of Recursion

Recursive code can utilize different types of recursion. One major distinction is made between *direct recursion* and *indirect recursion*. Indirect recursion requires at least two functions that call themselves mutually, whereas direct recursion needs only one function that directly calls itself within its own body [14]. In this study, we exclusively consider direct recursion since it is mostly used in simple code snippets and introductory computer science courses. Direct recursion can further be split into the following subgroups [14]:

- **Linear Recursion**
  A *linear recursive* function is a recursive function that calls itself no more than once in its body [39]. A particular case of linear recursion is tail recursion. A *tail recursive* function is a recursive function in which the last action of a method is a recursive call [39]. Listing 2.1 shows a linear recursive function that is not tail recursive, while Listing 2.2 shows a linear recursive function that is also tail recursive.

Listing 2.1: Linear Recursive Function, Which Is Not Tail Recursive

```
static int a(int n) {
    if (n <= 1)
        return 1;

    return n * a(n - 1);
}
```

Listing 2.2: Linear Recursive Function, Which Is Also Tail Recursive

```
static int b(int n, int result) {
    if (n <= 1)
        return result *1;

    return b(n - 1, result * n);
}
```

- **Tree Recursion**
  A tree recursive function is a function that calls itself at least twice [39].

Listing 2.3: A Tree Recursive Function

```
static int c(int n) {
    if (n <= 1)
        return 1;

    return c(n-1) + c(n-1);
}
```

- **Nested Recursion**
  A nested recursive function is a function where one of the arguments to the recursive function is the function itself.

Listing 2.4: A Nested Recursive Function

```
static int d(int n) {
    if (n <= 1)
        return 1;

    return n * d(d(n-1));
}
```

Given the varying degrees of performance and simplicity, it is plausible that different types of recursion distinctly impact comprehension processes [18]. Consequently, categorizing and comparing results based on recursion type is essential to identify and analyze potential differences.

2.2 Cognitive Load Theory

Cognitive Load Theory (CLT) is a psychological theory established by educational psychologist John Sweller in the late 1980s [41]. Based on this foundational work, further research examined the constraints of human working memory and how instructional design can be refined to improve learning by addressing these limitations [7].

The cognitive load measures are an important developer-centric indicator in software engineering [13]. Whereas in software engineering, there is no specific definition, *cognitive load* generally refers to the mental effort users spend while reading software artifacts or cognitive processing tasks [13]. Previous research examines the difficulties encountered by programmers in managing complex programming jobs, emphasizing that heightened complexity in recursion adds to cognitive load [13].

2.3 Program Comprehension

Currently, there are so many studies on *program comprehension* that it can be easily overlooked what those studies measure when they speak about the process of comprehension. This confusion is especially relevant since we cannot directly observe how well somebody understands code. In 2023, Wyrich provided a contemporary definition for the term program comprehension. In our study, we will follow his definition and, therefore, refer to program comprehension as "a person's intentional act and degree of accomplishment in inferring the meaning of source code."

2.3.1 Top-Down vs. Bottom-Up Comprehension

Program comprehension uses a mixture of two comprehension processes: *top-down comprehension* and *bottom-up comprehension*. Top-down processes typically happen when the programmer has knowledge about the domain of a program. Bottom-up processes happen when the programmer has insufficient domain knowledge and lead to comprehension — as indicated by the name — from the bottom up. That means processing the code character-by-character until the developer can integrate them into higher-level abstractions [22].

For this experiment, we aimed to prevent mainly taking advantage of knowledge about the domain of a program. Therefore, we enforced bottom-up comprehension within the code snippets.

2.4 Threats to Validity in Program Comprehension Studies

In Chapter 1, we already mentioned that our experiment offers the option to improve the validity of future program comprehension experiments. Thus, we clarify the term *threats to validity*. We use the classification from Marvin Wyrich, Justus Bogner, and Stefan Wagner [48]. According to their research, threats to validity can be split into four subgroups:

1. **Internal validity**
   refers to the extent to which the treatment or independent variable(s) were actually responsible for the effects seen on the dependent variable, e.g., threatened by technical issues

2. **External validity**
   refers to the degree to which the findings of the study can be generalized to other participant populations or settings, e.g., threatened the knowledge of participants, such as programming experience

3. **Construct validity**
   refers to the degree to which the operationalization of the measures in a study actually

represents the constructs in the real world, e.g., threatened by an experiment-specific task description

4. **Conclusion validity**
   refers to whether the conclusions reached in a study are correct, e.g. threatened by data inconsistency

In Chapter 5, we will discuss threats to internal, external, construct, and conclusion validity regarding our experiment.

2.5 Eye-Tracking Fundamentals

Eye-tracking is referred to as the collection of eye movement data. It is an effective tool to collect evidence regarding the cognitive processes and visual attention of participants [36]. This section gives an overview of the fundamentals of eye-tracking.

2.5.1 History

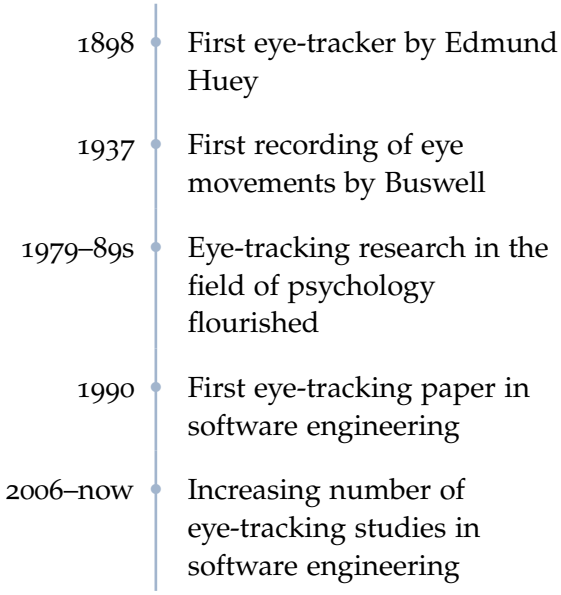
Traditionally, eye tracking has been used in cognitive psychology to study information processing tasks [30]. The first study in software engineering history using eye-tracking as a measuring tool was published in 1990 by Crosby et al. [36]. They studied program comprehension of procedural code and used the eye-tracker to discover the reading strategies of participants. While eye-tracking was expensive and complicated to use back then, over the years, eye trackers became more versatile, easy to use, and cheaper devices with satisfying software support, which led to increased recognition in software engineering since 2006 [35]. From that year on, it was established as a standard tool to study comprehension, debugging, collaborative interaction, and traceability [36]. Figure 2.1 overviews important dates in eye-tracking history.

2.5.2 Assumptions

Eye-tracking is based on two assumptions [17]: the *immediacy assumption* and the *eye-mind assumption*. The immediacy assumption implies that as soon as a participant sees a stimulus, they try to interpret it. A stimulus can be any object; in our case, it is the particularly presented source code snippet. The eye-mind assumption implies that a participant fixates their attention on a stimulus until they understand it.

With the help of those two assumptions, we can conclude that cognitive processes guide the visual attention of a participant to specific locations. Therefore, eye-tracking gives us the possibility to study those cognitive processes and effort while performing tasks such as comprehending code [35]. According to research by Obaidellah et al. and Sharafi et al. it helps researchers determine (1) why participants have problems finishing a task, (2) where participants expect to find specific elements, (3) whether elements are distracting, (4) how efficiently a design, layout, or artifact guides participants through a task, (5) whether there

FIGURE 2.1 Important Years for Eye-Tracking Research in the Field of Software Engineering[35]



are differences in the efficiency of a participant, based on their demographics or expertise, and (6) whether participants focus on details or briefly scanned the stimuli [35].

2.5.3 Eye-movement Classification

Eye movements are classified with the following terms, based on prior research [10, 15, 30, 36]:

- **Fixation:**
  A *fixation* is the stabilization of the eye on a part of the stimulus for a period of time for a longer [36]. The fixation data is provided by a time stamp and x and y coordinates. Research in the field of psychology shows that during fixation, the most cognitive processes and information acquisitions occur [10][15]. Readers primarily extract detailed information from the word they are directly fixating on and their following word. [30].
- **Saccade:**
  A *saccade* is the fast movement between fixations, usually for a period of time between 40-50 milliseconds [36]. According to research in the field of psychology, during saccades, information acquisition, and cognitive processing are very limited [10, 15].
- **Scan-path:**
  A *scan path* is a series of fixations on specific areas in chronological order [36].

This classification focuses on fixation, saccade, and scan-path as they are most relevant to the current context. Other types of eye movements, such as smooth pursuit or vestibulo-ocular movements, exist but are beyond the scope of this experiment.

Over the last decades, research in the field of psychology has proposed a distinction between two visual processing systems [15]:

- **Ambient Mode**
  The *ambient mode* is characterized by short fixations and long saccades and typically occurs during the initial stages of viewing. During ambient mode, individuals rapidly scan the environment to identify key areas of interest to efficiently navigate and focus on specific elements in subsequent viewing phases.

- **Focal Mode**
  The *focal mode* is classified as a period of time in which longer fixations and shorter saccades take place. It is capable of object identification and semantic categorization. This period usually appears during later viewing periods.

2.6 Eye-Tracking Data Analysis

Interpreting eye-tracking data in domains such as software engineering, specifically in program comprehension, is a challenging undertaking because of the large volume of raw data and various levels of abstraction required for helpful analysis. In order to simplify this issue, eye-tracking data can be classified into four distinct levels: first, second, third, and fourth-order data [35].

First-Order Data First-order data refers to the raw, unfiltered Eye-Tracking data, e.g., the x and y coordinates of each gaze point mapped to a location on the stimulus with their corresponding timestamp. Various studies also measure Pupil diameter and eye blinks as first-order data [35]. Since this type of data is inherently noisy and prone to environmental factors (such as lighting conditions) and therefore contains invalid data, it requires cleaning and preprocessing before analysis [40]. Common techniques include removing outliers and recalibrating erroneous fixations.

Second-Order Data Second-order data is derived from first-order data with the help of event detection algorithms. It involves the classification of eye movements into fixations and saccades based on spatial and temporal criteria, which is crucial to investigating the cognitive processes during visual tasks [35].

Third- and Fourth-Order Data Third- and fourth-order data is obtained by analyzing second-order data. From the classification of eye movements into fixations and saccades, various quantitative metrics can be derived to assess visual attention and cognitive processing during tasks [35, 36, 49]. These metrics are classified as third-order data if they concern quantifying fixations or saccades, or fourth-order data if they describe the sequence of these events [43]. Key third-order metrics include:

- **Fixation Count** This metric represents the total number of fixations on the stimulus recorded during the task. In the context of program comprehension, a higher fixation count on certain code segments may indicate areas where participants allocate more attention, possibly due to complexity or difficulty [35].

- **Mean Fixation Duration** The mean fixation duration refers to the length of time, typically measured in milliseconds, that the eye remains stationary during a fixation. Longer mean fixation durations can be associated with deeper cognitive processing or difficulty in interpreting the information being viewed. In software engineering, prolonged fixation durations on code elements may reflect the cognitive effort required to understand complex logic or unfamiliar syntax [35].
- **Fixations per Second** Also known as fixation frequency, this metric is calculated by dividing the total number of fixations by the total time spent on the task, yielding an average rate of fixations per second. A higher fixation rate per second may indicate a more active search strategy or increased effort in processing visual information [49].

Fourth-order data represents the highest level of abstraction in eye-tracking analysis. It describes the general reading order and search strategy and is also known as *scan paths* [43]. One way to visualize a scan path is shown in Figure 2.2.

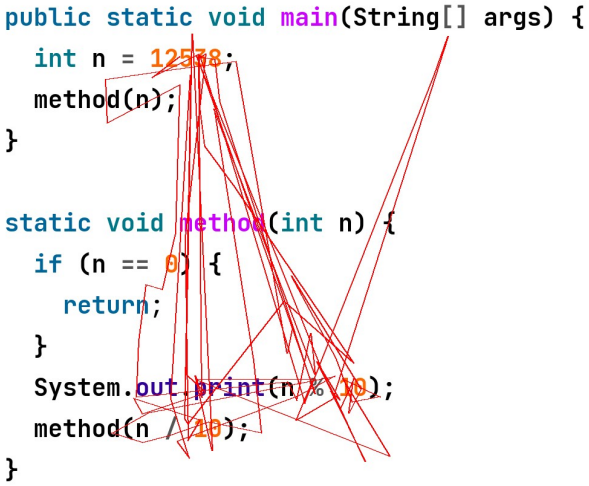


Figure 2.2: Example Scan Path for a Short Code Snippet

3 Methodology

This chapter describes the *methodology* of the thesis core evaluation, outlining the analysis goal, research questions, and experiment design.

3.1 Research Questions

RQ1: How does the input size affect students' response times and correctness when understanding recursive functions?

RQ2: How does input size affect students' visual attention when understanding recursive functions?

RQ3: Do different types of recursion affect the effect of input size on students' response times and correctness when understanding recursive functions?

RQ4: Do different types of recursion affect the effect of input size on students' visual attention when understanding recursive functions?

By proposing those research questions, we investigate to what extent input size affects the results of a program comprehension study and, therefore, aim to support the determination of a suitable input size for future code snippets. Moreover, the analysis of aspects other than input size in following program comprehension studies can be isolated better by abstracting input size as a source of irritation.

As elaborated in Section 2.5, eye tracking is an additional tool to give us insight into the cognitive processes of participants since those guide visual attention to specific locations. It allows us to study effort while performing the given task and to gather information about the intentions of participants. Furthermore, it can tell us why participants have problems executing a task, where to find specific elements, and whether they are distracting factors [35].

3.2 Independent Variables

Our measurements depend on multiple factors:

1. **Type of recursion:**
   The snippets were categorized by their applied *recursion type* since we expected different results for each group. We differentiated between linear recursion, tail

recursion, binary tree recursion, and nested recursion. Their differences were included in our evaluation, and the results are compared among and between each group.

2. **Input Size:**
   Each snippet was evaluated across four distinct input sizes to assess performance systematically under varying computational complexities. Specifically, for each snippet, we selected four input sizes to ensure that the recursive tree of the methods exhibited the following depths:
   - **Input Size 1**: Recursive tree depth of 3
   - **Input Size 2**: Recursive tree depth of 4
   - **Input Size 3**: Recursive tree depth of 5
   - **Input Size 4**: Recursive tree depth of 6

   Additionally, regarding tree recursion, one particular snippet (Snippet B.15) was designed to start with a recursive tree depth of 2 for input size 1, incrementing to a depth of 3 for input size 2, and so on. This adjustment was made to keep the snippet sufficiently simple, facilitating easier problem-solving while still providing meaningful data.

   We determined a recursive tree depth of three as a minimum so that participants spent at least some time on comprehension and a recursive tree depth of six as a maximum so that the task was still solvable without pen and paper. In Section 3.5, we discuss more snippet details.

3.3 Setup

Figure 3.1: Tobii Eyex Tracker[1]

Using an *eye tracker* to answer our research questions required an *in-person lab experiment*. Participants were presented with 15 different code snippets that varied in terms of recursion type and input size. The study involved within-subject factors, as each participant experienced all levels of recursion types and input sizes across the snippets. However, although the same snippets were shown to all participants, the input size for each snippet varied. This variation makes the combination of input size and snippets a between-subject factor. Therefore, the study employs a *mixed design* that combines both within-subject and between-subject comparisons.

Since the participants could recognize what a snippet does in advance, we chose not to show the code with unspecific input beforehand. Instead, the input was provided by directly assigning a value to a variable in the code line.

For our experiment, we used the *Tobii Eyex*[1] *eye tracking device* pictured in 3.1. The choice of eye tracker can make a crucial difference in the resulting data [9]. Given that over half of the research conducted between 1990 and 2023 included a Tobii eye tracker, our selection is both comparable and facilitates simpler replication. The Tobii Eyex belongs to the newer generation of non-intrusive eye trackers. Compared to an intrusive eye tracker, it is video-based and works with Backlight Assisted Near Infrared Technology. The computer tracks eye movements by evaluating the positions of the participant's head using eyebrows, noses and lips, corneal reflection, and pupil center [36].

With a tracking population of 95%, frequency of 70 Hz, and 0.5-degree accuracy, the Tobii EyeX delivers reliable results. Before the experiment, the eye tracker was calibrated for each participant using the software provided by the manufacturer. We placed the participants at an ideal distance of 50-60 cm in front of a 2560x1440 screen. Visual distractions were limited to avoid participants looking at other stimuli and consequently contaminating eye-movement data. A custom C# program previously used for several other experiments [2, 29] guided the participants through the calibration process, demographic questions, task, and code snippets while capturing response time, answers and eye movements.

3.4 Execution

3.4.1 Pre-Study

To gain a sense of how long the provided task takes to solve for each snippet and to find out how many snippets are needed for our experiment, we conducted a *pre-study* in which we presented several example snippets to three computer science undergrad students. On average, it took them roughly 90-120 s to solve a single task.

According to previous research in the field of psychology, an eye-tracking session should last no longer than 90 minutes due to changes in physiology, e.g., dryness caused by fatigue [35]. The general rule is that if a session is longer than 30 minutes, participants should be given time to relax their eyes between successive stimuli [35]. This will prevent the overexhaustion of participants, which will lead to unreliable results. Considering our pre-study with about 90-120 seconds of processing time, we were able to present fifteen code snippets per person in the given time frame.

3.4.2 Task

The participants were tasked with determining the output of their print statements for a total of 15 different code snippets.

1 https://help.tobii.com/hc/en-us/articles/212818309-Specifications-for-EyeX

Presentation and Distribution of Snippets After an initial warm-up task (see B.16, participants reviewed the 15 code snippets consecutively. Each snippet was presented with one of the four distinct input sizes (ranging from 1 to 4), ensuring that every participant received an input size from 1 to 4 for each snippet. This method guaranteed that each participant encountered a mix of simple and challenging inputs, balancing the cognitive load throughout the session. The detailed order in which each code snippet was presented is listed in the Appendix (see Table B.17).

Data Collection The snippets were distributed in such a way that, after 12 participants, we accumulated three data points for each snippet-input combination. This distribution strategy provided robust and reliable data for each scenario, facilitating a thorough analysis of how input size influences the cognitive processes involved in mentally executing recursive calls.

During the presentation, participants were prohibited from writing down or altering their initial answers, as the study aimed to investigate the cognitive processes involved in mentally executing recursive calls without external aids. The entire process is visualized in Figure 3.2.

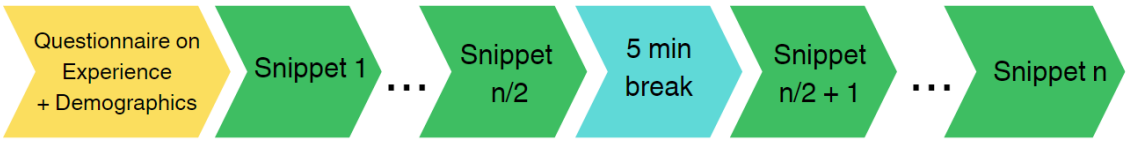


Figure 3.2: Visualization of the Task with N Code Snippets

3.4.3 Participants

To provide an initial insight into our topic, a small amount of twelve *participants* were gathered to take part in our main experiment. Recruiting more participants meant there would be more gathered data, but the study would also be challenging in the constrained time frame of a bachelor's thesis.
Every participant needed to have a similar level of programming expertise, as varying levels of experience can result in differences in accuracy, response times, and eye movements when comprehending recursive programs [28]. Hence, variation in experience makes results between subjects less comparable. Furthermore, participants needed to be able to understand simple code snippets and algorithms. An appropriate sample consisted of students who were actively enrolled in the Bachelor of Computer Science or an equivalent program throughout the duration of the study. Students, in general, are relatively easy to recruit for research purposes and possess sufficient knowledge to participate in the experiment. In addition, the criteria were validated by exclusively selecting individuals for our study who had successfully completed the "Programming 2" course.

To check that the participants were suited for the study and to add supplementary data to put the results into perspective, they completed a *pre-experimental questionnaire* on

demographics and programming experience. Summarized, eight out of twelve participants were male, and four were female. Since the male/female ratio in computer science programs at our university is roughly 75/25, our participant rate is close to representing this gender distribution. All of the participants had achieved a High School Diploma and were between 21 and 23 years old. Regarding Java knowledge, most of them have reported project experience (7), followed by basic knowledge (3) and regular knowledge (2), with an average of 1.5 years of Java experience and an average of four years of total programming experience. Figure 3.3 visualizes the age distribution, java experience, and total programming experience. Beyond two participants wearing glasses, no further eye problems were classified.

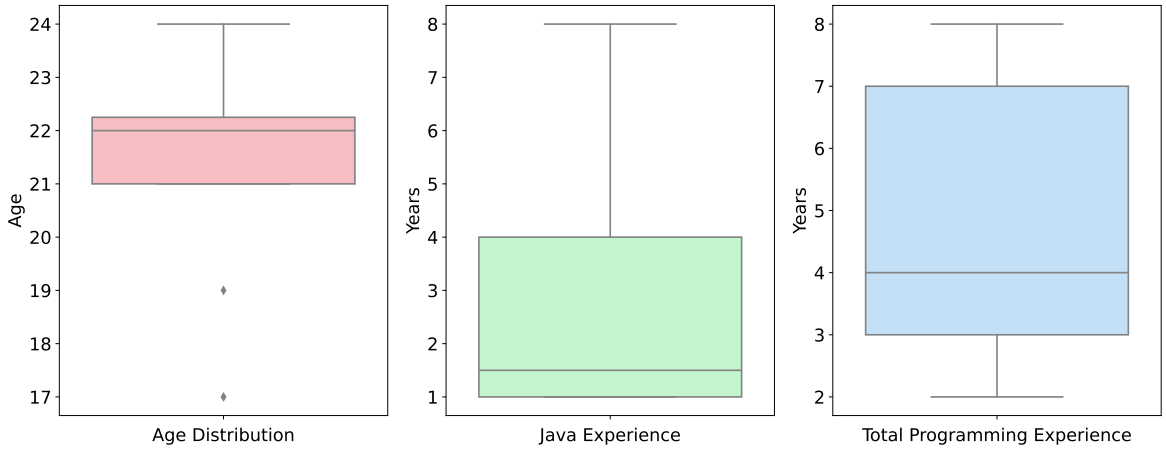


Figure 3.3: Visualization of Participants' Age and Programming Experience

3.4.4 Post-Questionnaire

Following the primary experiment, the participants were required to answer a *post-questionnaire* during an interview. It was conducted to obtain further insights and observations to aid in answering the proposed research questions. We aimed to gather information on the perceived difficulties of participants' problem-solving tactics and gaze strategies while considering the recursive depth of the snippets. The questions are listed in Table 3.1.

3.4.5 Deviations

During the course of the study, we encountered technical issues that resulted in the loss of two eye-tracking data points. This affects one out of three planned data points of Snippet B.1 with input size 4 and one out of three planned data points Snippet B.5 with input size 4.

Furthermore, during the evaluation process, code snippet B.14 was excluded to maintain the integrity and consistency of our study. This exclusion was necessary because the versions of the task corresponding to input sizes 2, 3, and 4 inadvertently included an additional line of code that was absent in the input size 1 version.

Table 3.1: Questions Asked In the Post-questionnaire of the Primary Experiment

Main Question	**Subquestions**
How did you find the study?	Please describe your experience with the code snippets. Which snippets felt hard and which felt easy to you? What other difficulties did you experience?
Our main goal is to study the effect of input sizes and depth on visual attention, correctness, and response time while understanding recursive functions. Did the input size or recursive depth alter the difficulty of your output calculation for the task?	How did the input size or recursive depth of the functions affect response time and correctness while solving the task?
Regarding input size and recursive depth, describe your thought process to find the output.	Did you try to understand the general functionality of the code first, or did you go through the code step by step? Did that differ with input size or recursive depth?
How did the input size or recursive depth affect gaze strategies and the line reading order?	Did you adopt a specific gaze strategy while navigating through the code snippets? Did that differ with the input size or recursive depth?
Share your general impressions.	Do you have any additional observations, insights, or thoughts regarding the input size or depth of recursive functions in program comprehension that could be relevant to our findings?

3.5 Code Snippets

Using *code snippets* from existing studies is powerful to compare results and prove their validity [48]. Marvin Wyrich, Justus Bogner, and Stefan Wagner delivered eligible code snippets already used in one of their program comprehension experiments. In addition to adapting those, we expanded the collection with our snippets, respecting homogeneity concerning potential confounders. Table 3.2 lists all collected snippets.

3.5.1 Criteria

We propose the following *criteria for consistency* oriented on the collection of criteria from Marvyn Wyrich, Justus Bogner, and Stefan Wagner [48]:

- **Written in Java**
  Java is an ideal language for our experiment since it is widely spread and the most used programming language in program comprehension studies with or without eye tracking [36, 48].
- **Simple number as input**
  Simple numbers given as integer or string input are the plainest inputs for answering our research questions. Reducing the inputs to only one type makes the code snippets simple and consistent.
- **Novice friendliness**
  Most of our ideas came from everyday beginner programming tasks, typical textbook problems, or algorithms known to novice programmers.
- **Simplicity**
  The snippets needed to be small enough to be understood in under two minutes but large enough to require some effort. Furthermore, they must fit on a single screen to avoid scrolling, which our eye-tracking evaluation strategy does not support.
- **Minimization of unrelated factors**
  The influence of unrelated factors, such as naming styles or formatting, had to be minimized. Consequently, those factors were made consistent.

3.5.2 Enforcing Bottom-Up Comprehension

As Section 2.3.1 explained, the snippets were adjusted for *bottom-up comprehension*. Hence, it was important to force participants to comprehend the code statement by statement. One confirmed way to do this is by obscuring identifier names [29]. Rather than selecting meaningful names that indicate their intended usage, the code snippets use alphabetically sorted meaningless character names for variables and method names.

The Following snippets are examples that fulfill the snippet criteria:

Listing 3.1: Example Code Fragment Tree Recursion: Calculation of the nth Fibonacci Number

```
public static void main(String[] args) {
    int n = 3;
    System.out.println(method(n));
}

static int method(int n) {
    if (n <= 1)
        return n;
    return method(n-1) + method(n-2);
}
```

Listing 3.2: Example code fragment Linear Recursion: Calculation of the faculty of n

```
public static void main(String[] args) {
    int n = 3;
    System.out.println(method(n));
}

static int method(int n) {
    if (n == 0)
        return 1;
    return n * method(n-1);
}
```

3.6 Dependent Variables

The *dependent variables* are:

1. **Response correctness**:
   Since the research focus of the experiment, which answers the research questions in Section 3.1, is the effect of input size on program comprehension, it is essential to focus on semantic correctness to determine if comprehension took place. Following this, the answers of participants were evaluated manually.

2. **Response time**:
   The period considered as response time begins with the participant seeing the respected source code snippet for the first time and ends with the participant entering the result.

3. **Eye movements**:
   Based on the immediacy and eye-mind assumptions, as explained in Section 2.5.2, participants fixate their attention on a stimulus until they understand it. Therefore, eye tracking is a great tool to measure cognitive load and visual attention to answer the proposed research questions in Section 3.1.

Table 3.2: Snippet Overview Grouped by Type of Recursion and Their Subject

Type of recursion	Subject	LoC
Linear Recursion	*Factorial*	10
	Decimal to binary	11
	Triangular number	10
	Sum of digits	10
	Logarithm	11
Linear Recursion: Tail Recursion	*Prime factors*	13
	Reverse Numbers	11
	Is number a Palindrome?	17
	Countdown	10
	Test if 2 digits sum up to 5	18
Tree Recursion	*Fibonacci*	10
	Modified towers of Hanoi	12
	Sequence 1	12
	Biggest Number	20
	Sequence 2	14
Nested Recursion (warm-up snippet)	*Calculation*	10

3.7 Evaluation

This chapter presents our *data analysis methodology*.

Eye Tracking Data Analysis Our custom C# program gathers the raw and unfiltered eye-tracking data. This so-called *first-order data* (see Section 2.6), in our case, contains the x and y coordinates of each gaze point mapped to a location on the stimulus with their corresponding time stamp. After preprocessing the data to remove invalid data points and noise, fixation and saccade data are derived from the cleaned dataset (so-called *second-order data*). To evaluate visual attention and cognitive processes, we analyze the data using several key *third-order* metrics. These include Fixation Count, Mean Fixation Duration, and Fixation Count per Second. We chose *Fixation Count* as a relevant metric since a higher number of fixations on a stimulus indicates that more attention is required to process the stimulus and, therefore, gives us insight into the task's complexity and difficulty [35]. The same applies to the *Mean Fixation Duration* since fixation durations reflect the cognitive effort needed to comprehend logical problems [35]. Furthermore, we analyzed the *Fixation Count per Second* because a higher fixation rate per second can indicate a more active search strategy and increased effort in processing visual information [49]. The metrics are detailed in the Background (see Section 2.6).

Statistics To determine the significance of our results, we applied several *statistical tests* based on our data. We employed a standard *significance level* of 0.05 for the tests to balance the risk of false positives while ensuring that the findings are statistically meaningful and not due to random chance.

Shapiro-Wilk: The Shapiro-Wilk test is a statistical test used to assess the normality of our data [33]. We employed the Shapiro-Wilk test, due to its high power and effectiveness for the analysis of small sample sizes, to verify that our data is not normally distributed, thereby ensuring that our subsequent statistical analyses are appropriately chosen based on the underlying data characteristics. A standard *significance level* of 0.05 was used to reject the null hypothesis of normality. The results were calculated using the scipy python package [45].

Kruskal-Wallis: The Kruskal-Wallis test is a non-parametric statistical test used to determine if there are statistically significant differences between two or more groups on a particular metric when the data is not normally distributed and ordinal [19]. We chose this test for our first and second research questions since we have four groups of input sizes to compare, and our data is not normally distributed but ordinal. The results were calculated with the scipy python package [45].

Dunns: The Dunn's test is suitable as a bi-directional post-hoc test after the Kruskal-Wallis test to discover which groups are significantly different from one other [11]. The Dunn's test compares each pair of groups and returns p-values for the differences between them. We applied the False Discovery Rate (FDR) correction to adjust for multiple comparisons, controlling the expected proportion of false discoveries [4]. The results were calculated with the scikit_posthocs python package [42].

Two-way ANOVA: The two-way ANOVA is used to evaluate the effect of two independent categorical variables on a continuous dependent variable [25]. It helps determine whether each independent variable significantly influences the dependent variable on its own and whether there is a significant interaction between the two independent variables. We chose this test for our third and fourth research questions to find interaction effects between the type of recursion and input size on the collected gaze data. The results were calculated with the statsmodels python package [38].

Tukey HSD: In the context of a two-way ANOVA, if significant main effects or interactions are detected, the Tukey HSD test can be employed as a post-hoc analysis to identify which specific pairs of group means are significantly different from each other [44]. The results were calculated with the statsmodels python package [38].

Cliff's Delta: The Cliff's Delta test is a non-parametric effect size measure that quantifies the magnitude of difference between two groups. It represents the probability that a randomly selected value from one group is higher than a randomly selected value from the other, minus the reverse probability [6]. This measure is particularly useful for ordinal or non-normally distributed data, providing an intuitive interpretation of the practical significance of the results. We employed Cliff's Delta to complement the p-values obtained from the Dunns and Tukey HSD tests, thereby assessing the statistical significance and the effect size of the differences observed. The results were calculated using the cliffsDelta Python package [12].

4

Results

This chapter reveals the *results* of the thesis and addresses the *research questions* outlined in Section 3.1. To answer these research questions, we collected data on response time, accuracy, and the x and y coordinates of eye movements, accompanied by their corresponding timestamps. The sections in this chapter are organized according to the four research questions:

Section 4.1: The first section presents data addressing RQ1, focusing on the effect of input size on response times and correctness.

Section 4.2: The second section addresses RQ2, investigating how input size affects the visual attention of novice programmers using eye-tracking data.

Section 4.3: The third section combines input size and recursion type to explore their joint effect on response times and correctness (RQ3).

Section 4.4: The final section examines the interaction of input size and recursion type on the visual attention of novice programmers (RQ4) based on eye-tracking metrics.

While we focus on the research questions specifically in those sections, the complete results on task performance and gaze data can be found in the Appendix (see Table A.1 and Table A.2).

When discussing snippets with an input size of 1, we refer to those with a recursive tree depth of three. For snippets with an input size of 2, the recursive tree depth is four. For an input size of 3, the recursive tree depth increases to five, and for an input size of 4, it reaches a depth of six. For more details on the selected input sizes and exceptions, visit Section 3.4.

4.1 Effect of Input Size on Correctness and Response Time

This section presents the results based on the *behavioral data* gathered during the experiment to answer how the input size affects the response times and correctness of novice programmers when understanding recursive functions, hence addressing our *first research question*.

4.1.1 Descriptive Statistics

The *descriptive statistics* provide an overview of general trends in our behavioral data grouped by the task input size. We collected data from a total of 180 task instances, with an average of 45 data points for each input size.

Correctness The proportion of accurate responses diminished as the input size expanded, indicating a consistent trend of lower performance with bigger inputs. For the minimal input size (1), participants responded accurately 81% of the time. Nonetheless, as the input size increased, accuracy diminished to 66.7% for an input size of 2 and further declined to 52.4% for an input size of 3. Notably, at an input size of 4, accuracy increased marginally to 57.1%. However, it continued to be considerably lower than the accuracy for the smallest input size. We show the detailed results in Table 4.1 and visualize them in Figure 4.1.

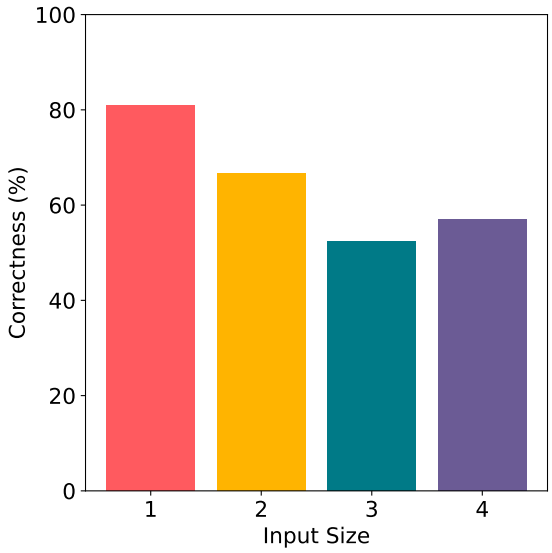


Figure 4.1: Correctness by Input Size

Response Time As input size increased, participants took longer to respond, with a noticeable increase in the variability of response times. At input size 1, the mean response time was 49.2 seconds, with a standard deviation of 26.0 seconds, indicating relatively consistent performance. However, as input size increased, response times grew to 62.7 seconds for input size 2 and further to 76.0 and 99.9 seconds for input sizes 3 and 4, respectively. The variability in response times measured by the standard deviation also increased with input size, suggesting that the strategies of participants and their ability to cope with the complexity of the recursive functions became more varied as the problem grew more complicated. Interestingly, the mean correct-only response times were generally lower than the overall response times, particularly for smaller input sizes. Correct responses were achieved more quickly for input sizes 1 and 2, with mean times of 36.8 seconds and 34.4 seconds, respectively. However, for input sizes 3 and 4, the mean correct-only response

times showed greater variability, with input size 4 showing a higher time of 44.8 seconds and the greatest standard deviation (57.4 seconds). We show the mean response times, the mean response times for correctly answered tasks, and standard deviations in Table 4.1. Furthermore, we visualize the response times in Figure 4.2 and the response times for correctly answered tasks only in Figure 4.3.

Table 4.1: Correctness Percentages and Statistics Related to Response Times for Different Input Sizes

Input	Correctness (%)	Time (s) ± SD	Correct Only Time (s) ± SD
1	81.0	49.2 ± 26.0	36.8 ± 28.9
2	66.7	62.7 ± 42.2	34.4 ± 34.6
3	52.4	76.0 ± 60.2	27.9 ± 32.9
4	57.1	99.9 ± 84.7	44.8 ± 57.4

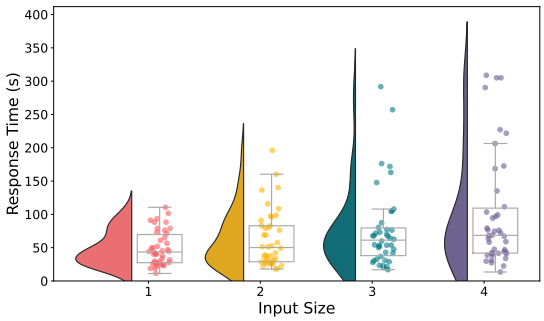


Figure 4.2: Response Time for each Input Size

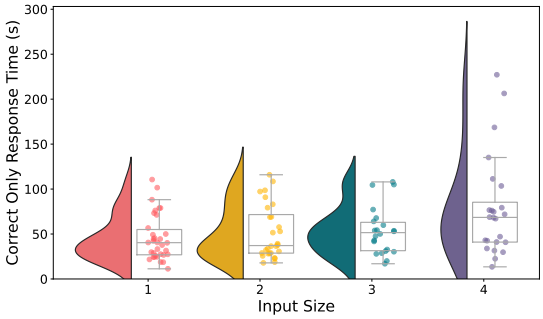


Figure 4.3: Correct Only Response Time for each Input Size

4.1.2 Statistical Tests

Kruskal-Wallis The Kruskal-Wallis test shows whether there are statistically significant differences in correctness and response time between the different input sizes [19]. The p-value for correctness (0.034) is below the chosen significance threshold of 0.05, indicating a statistically significant difference in correctness among the groups based on input size. Likewise, the p-value for response time (0.009) is below 0.05, indicating a significant difference in response time among the groups. Table 4.2 outlines the statistical test results for correctness and response time.

Dunn's Since the Kruskal-Wallis test indicated significant differences, we used the Dunn's test with *False Discovery Rate* (FDR) correction as a *post-hoc* test to determine which input sizes lead to significantly different results from each other. The pairwise comparison between functions with input size 1 and size 3 shows a difference close to significance

Table 4.2: Kruskal-Wallis Test Results for Behavioral Data

Metric	Statistic	P-Value
Correctness	8.659	**0.034**
Response Time	11.467	**0.009**

in correctness with a p-value of 0.094; however, the difference remains insignificant. The other comparisons indicate non-significant differences (p-values > 0.05), implying that the correctness between these groups is not statistically different. We list the complete results for correctness in Table 4.3.

Dunn's test for response time provides the p-values between the different input sizes listed in Table 4.4. The comparison between functions with input size 1 and input size 4 shows a significant difference in time with a p-value of 0.018. Regarding the descriptive statistics, we can guess that this indicates that participants took significantly longer for input size 4 compared to input size 1. Additionally, Cliff's delta analysis for this comparison yielded a medium effect size of -0.402, suggesting a moderate practical significance of this difference. Other comparisons show no significant differences. We list the complete results for response time in Table 4.4.

Table 4.3: Dunn's Pairwise Comparison P-values for Correctness

Input	1	2	3	4
1	1.000			
2	0.636	1.000		
3	0.095	0.636	1.000	
4	0.170	1.000	1.000	1.000

Table 4.4: Dunn's Pairwise Comparison P-values for Response Time

Input	1	2	3	4
1	1.000			
2	0.793	1.000		
3	0.179	0.793	1.000	
4	**0.018**	0.179	0.793	1.000

4.1.3 Discussion

Our analysis reveals a clear impact of input size on the correctness and response times of novice programmers when understanding recursive functions.

Correctness The descriptive statistics show a decline in correctness as the input size rises. This decline could be attributed to the increased cognitive load associated with processing more complex recursive calls and deeper recursion levels. The Kruskal-Wallis test confirmed a statistically significant difference in correctness among different input sizes. However, the Dunn's post-hoc test did not identify significant differences between specific input size pairs. Therefore, we did not calculate effect sizes. This suggests that while the overall effect of input size on correctness is significant, the differences between individual input sizes may

be subtle or inconsistent, potentially requiring further investigation with a larger sample size or more detailed analytical methods.

Response Time Response times consistently increased with input size, and significant differences with a medium effect size were found between sizes 1 and 4. The combination of statistical significance and a medium effect size underscores that the increase in response time is not only statistically detectable but also practically meaningful. Furthermore, increasing standard deviations with larger inputs suggest greater variability in how novice programmers cope with complexity, potentially reflecting differences in individual problem-solving strategies or levels of understanding.

The observed decline in correctness and increase in response time with larger input sizes align with the *cognitive load theory* (see Section 2.2), which posits that as the amount of information to be processed increases, so does the associated cognitive load. Recursive functions with larger inputs likely require participants to maintain more information in their working memory, leading to increased processing time and a higher likelihood of errors.

Our post-questionnaire results further support this observation. More than half of the participants reported that tasks became increasingly difficult as input sizes grew. Nearly all participants noted that with larger input sizes, they frequently jumped between lines of code, indicating a greater cognitive effort needed to trace recursive calls.

Furthermore, several participants mentioned that remembering the output and order of recursive calls posed significant challenges. Hence, this increased cognitive load with larger input sizes may be attributed to the restriction on note-taking, which led to participants depending more heavily on their memory.

We did not find a significant difference in difficulty between input sizes 1, 2, and 3. However, the medium effect size in response time between input sizes 1 and 4 indicates that the function having a recursive tree depth of 6 has a substantial practical impact on the performance of novice programmers. This presumably applies to larger input sizes as well when considering the trend in our descriptive statistics.

Speed-Accuracy Tradeoff Interestingly, the patterns observed in correctness and response time do not fully align. While both measures are influenced by cognitive load, the discrepancies suggest the presence of a speed-accuracy tradeoff. The speed-accuracy tradeoff (SAT) is a recognized principle in cognitive experimental studies and states that as the efficiency of information processing increases, response times will tend to decrease while accuracy increases [8]. Following this principle, prioritizing correctness can increase accuracy at the expense of longer response times, which may explain the variations observed in our participants' performance. However, there are additional possible explanations for these discrepancies. Discontinuities in cognitive processes, such as switching problem-solving strategies, can impact performance metrics differently [5]. For instance, participants may shift from line-by-line code tracing to attempting a holistic understanding of the recursive function as tasks become more complex, affecting correctness and response time in uneven ways. Furthermore, a decline in cognitive efficiency and increasing cognitive depletion over time—especially when working on problems that require high effort—could lead

to variations in accuracy and speed [5]. Prolonged engagement with complex tasks may exhaust cognitive resources, resulting in slower responses or increased errors independent of the speed-accuracy tradeoff.

Implications Previous research showed that teaching recursion is difficult [24]. This thesis demonstrates that the level of difficulty experienced by novice programmers is related to the size of the input they receive. The results emphasize the need for educators to scaffold learning experiences when teaching recursion. One idea is to gradually increase input sizes to build the confidence and understanding of novice programmers. Educators should be mindful of the challenges posed by larger input sizes in recursive functions, especially considering the jump in difficulty from input sizes 3 to 4. Introducing recursion with smaller inputs could help novice programmers grasp the fundamental concepts before progressing to more complex problems. Encouraging collaborative problem-solving and providing practice with incremental increases in input size can also support novice programmers in developing more effective strategies for understanding recursion.

Furthermore, future program comprehension experiments that use specific input sizes for recursive functions, as several experiments have done before (e.g. [1, 20, 28]), must take into account how the selected input size influences the results for correctness and response time. There are several studies that use an unspecific input n for recursive functions (e.g. [23, 46]). Researchers could improve control over their results and limit ambiguous interpretations by choosing a specific and sensible input size. This holds relevance for multiple reasons. Knowing the exact input sizes helps in optimizing recursive functions for particular scenarios. Furthermore, by standardizing the input size, researchers can isolate the effects of the recursive function itself without the variability introduced by changing input sizes and thereby enhance the conclusion validity of the study. It allows for more controlled experiments, meaningful comparisons, and results that are both reliable and relevant.

4.2 Effect of Input Size on Visual Attention

This section presents the results based on the *eye-tracking data* gathered during the experiment to answer how the input size affects the visual attention of participants when understanding recursive functions, hence addressing our *second research question*.

4.2.1 Descriptive Statistics

The *descriptive statistics* provide an overview of general trends in our eye-tracking data. As for correctness and response time, we gathered an average of 45 data points per input size, resulting in a total of 180 eye-tracking records. The following paragraphs focus on the mean results grouped by input size shown in Table 4.5.

Fixation Count and Saccade Count The mean fixation count and mean saccade count both show an upward trend as the input size progresses from 1 to 4. Specifically, the fixation

Table 4.5: Descriptive Statistics of Eye-Tracking Metrics by Input Size

Input	Mean Fixation Count ± SD	Mean Saccade Count ± SD	Mean Fixation Duration (s) ± SD	Mean Fixations per Second ± SD
1	182 ± 83.8	182 ± 83.9	0.206 ± 0.04	3.76 ± 0.62
2	219 ± 113.8	219 ± 113.7	0.216 ± 0.05	3.61 ± 0.58
3	266 ± 167.3	266 ± 167.3	0.230 ± 0.06	3.47 ± 0.59
4	293 ± 202.8	293 ± 202.7	0.238 ± 0.06	3.27 ± 0.57

and saccade count both increase from a mean of 182 for input size 1 to 293 for input size 4. They are identical because each fixation is directly associated with a corresponding saccade. This indicates that more complex tasks may require more visual exploration and cognitive processing. We visualize the trend in Figure 4.4.

Mean Fixation Duration The mean fixation time demonstrates an increasing trend, with a minimum duration of 0.206 seconds recorded in input size 1 and a maximum duration of 0.238 seconds in input size 4. This rise may suggest that participants require longer fixations to understand increasingly complicated tasks as they advance. The trend is visualized in Figure 4.5.

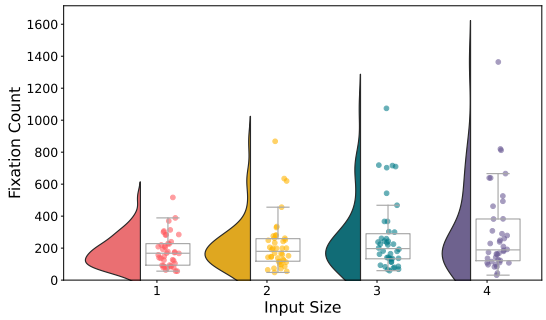


Figure 4.4: Fixation Count for each Input Size

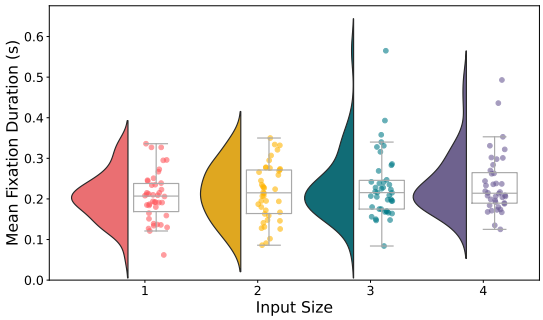


Figure 4.5: Mean Fixation Duration for each Input Size

Mean Fixation Count per Second The mean fixations per second exhibit a declining trend throughout input sizes, beginning at 3.76 for input size 1 and reducing to 3.27 for input size 4. This small reduction could imply that as tasks become more demanding, participants take longer fixations and produce fewer fixations per second, presumably representing higher cognitive load or processing time. Figure 4.6 visualizes the decline.

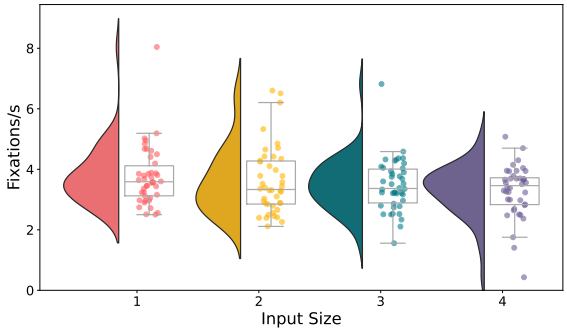


Figure 4.6: Mean Fixation Count per Second for each Input Size

Table 4.6: Kruskal-Wallis Test Results for Visual Attention Metrics

Metric	Statistic	P-Value
Fixation Count	3.526	0.317
Mean Fixation Duration	2.918	0.404
Fixation Count Per Second	3.135	0.371

4.2.2 Statistical Tests

Kruskal-Wallis We performed the Kruskal-Wallis test to show us the statistical significance of visual attention data between our four input sizes [19]. It showed no significant differences in the comparisons. Therefore, we did not perform a post-hoc test or calculate the effect sizes. The test results are outlined in Table 4.6.

4.2.3 Discussion

The descriptive statistics reveal observable trends that provide insights into the cognitive processes of novice programmers when dealing with recursive functions of varying complexity. As for correctness and response time, the observed trends in eye-tracking data align with the cognitive load theory (see Section 2.2). The increase in fixation count and mean fixation duration suggests that larger input sizes may impose a greater cognitive load on novice programmers as they attempt to simulate the recursive processes mentally. The decrease in fixations per second further supports the notion that novice programmers slow down their visual scanning to accommodate the increased processing requirements.

However, we must not draw firm conclusions from these results since the statistical analysis did not confirm any significant differences. Even if trends are present, they may not be large enough between our input sizes to reach statistical significance. But there is more than this potential reason for our result. A small sample size, as in our study, diminishes the power of a statistical test, making it less likely to detect a true effect even if one exists. This occurs since each data point has a higher impact on the overall statistics, and consequently,

their variability raises uncertainty. Furthermore, the occurrence of statistically significant results might be contingent upon the testing of inputs of a greater magnitude. In conclusion, even though our results for the second research question are not statistically relevant, does not mean that there is no relevant answer to the research question overall.

4.3 Effect of Type of Recursion on the Effect of Input Size on Response Time and Correctness

This section presents the results derived from the *behavioral data* collected throughout the experiment, exploring how the type of recursion influences the effect of input size on the accuracy and response time of novice programmers in comprehending recursive functions, thereby addressing our *third research question*.

4.3.1 Descriptive Statistics

The descriptive statistics offer an overview of general trends in our behavioral data grouped by the task input size and recursion type. For each combination of recursion type and input size, we collected data from an average of 15 answers, resulting in a total of 180 data points for this analysis. Table 4.7 lists the collected data.

Table 4.7: Mean Correctness and Response Time by Recursion Type and Input Size

Recursion	**Input**	**Mean Correctness** (%)	**Mean Response Time (s)** $\pm$ **SD**
Linear	1	80.0	42.0 ± 16.6
	2	86.7	49.6 ± 25.8
	3	66.7	47.3 ± 18.8
	4	66.7	87.5 ± 61.3
Tail	1	83.3	69.0 ± 30.4
	2	66.7	73.5 ± 41.7
	3	83.4	61.1 ± 17.9
	4	83.4	104.0 ± 35.8
Tree	1	80.0	40.6 ± 17.6
	2	46.7	67.1 ± 33.6
	3	13.3	116.7 ± 60.7
	4	26.7	108.9 ± 92.4

Correctness

- **Linear Recursion**: The correctness in linear recursion tasks begins at 80.0% for input size 1 and reaches a maximum of 86.7% for input size 2. Nonetheless, it diminishes for input sizes 3 and 4, both at 66.7%, suggesting that as the complexity of linear recursion rises, the correctness of participants tends to decline.
- **Linear Recursion - Tail Recursion**: Tail recursion shows a somewhat consistent performance. Correctness begins at 83.3% for input size 1, drops to 66.7% for input size 2, but increases to 83.4% for input size 3 and 4. Those results indicate that while some tasks (like those with input size 2) may challenge participants, tail recursion overall maintains a relatively stable level of correctness, even for more complex tasks.
- **Tree Recursion**: Tree recursion shows the most substantial decrease in correctness. It starts at 80.0% for input size 1 but drops rapidly to 46.7% for input size 2 and continues to fall to 13.3% and 26.7% for input sizes 3 and 4, respectively. Those results suggest that tree recursion is the most challenging for participants, leading to a sharp decline in accuracy as task complexity increases.

Figure 4.7 visualizes the progressions for correctness.

Response Time

- **Linear Recursion**: Response times gradually increase as the input size increases. The mean response time for recursive functions begins at 42.0 seconds with an input size of 1 and escalates to 87.5 seconds with an input size of 4. This suggests linear recursion may scale moderately in response time as task complexity grows.
- **Linear Recursion - Tail Recursion**: Tail recursion tasks exhibit higher response times than linear recursion tasks, starting at 69.0 seconds for input size 1. The mean response time peaks at 104.0 seconds for input size 4, indicating that tail recursion becomes more time-consuming with increasing task complexity.
- **Tree Recursion**: Tree recursion shows the most significant increase in response time, with a relatively quick response input size 1 (40.6 seconds). However, by input size 3, the mean response time jumps sharply to 116.7 seconds; for input size 4, it slightly decreases to 108.9 seconds. This pattern indicates that tree recursion tasks may pose the highest cognitive load, resulting in longer response times, especially for more complex tasks.

The progressions for response time are visualized in Figure 4.8.

Overall, the data indicates that both response time and correctness are affected by the type of recursion and task complexity, with tree recursion being particularly challenging in terms of both time and accuracy.

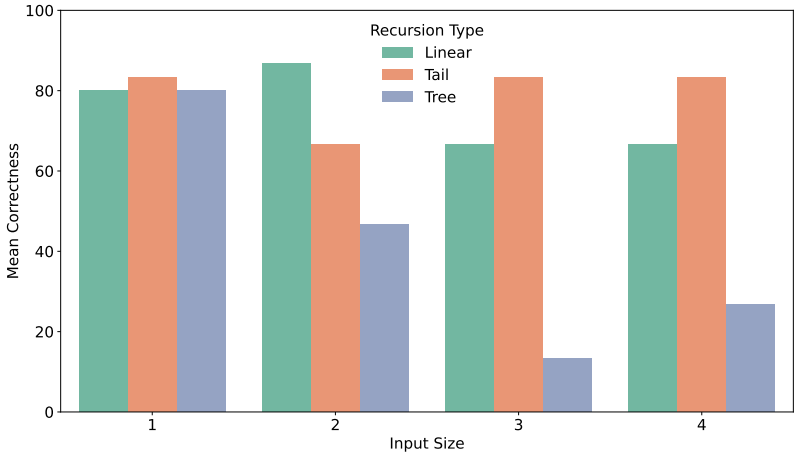


Figure 4.7: Correctness for each Input Size and Recursion Type

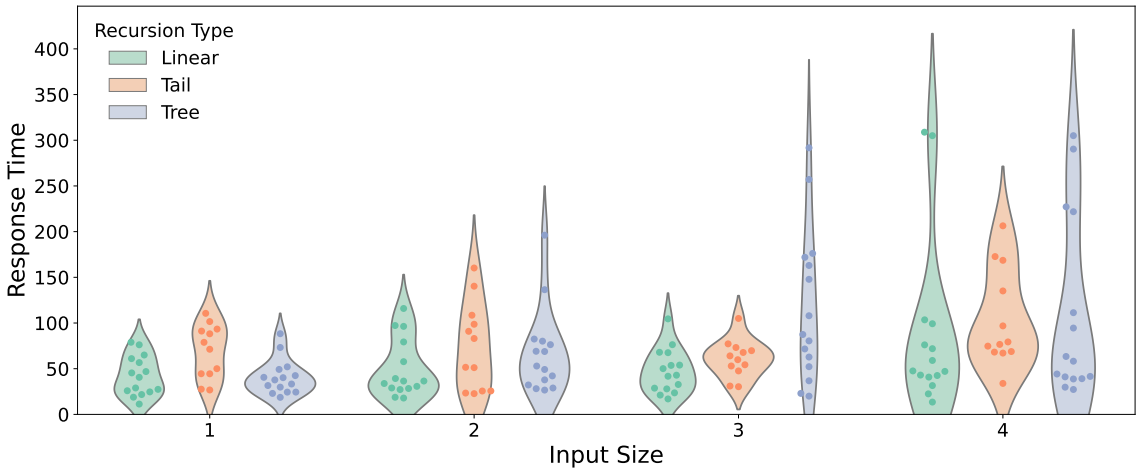


Figure 4.8: Response Time for each Input Size and Recursion Type

4.3.2 Statistical Tests

Two-Way ANOVA The two-way ANOVA test can show whether there is a *significant interaction* between input size and correctness and input size and response time.

- **Correctness** The two-way ANOVA shows that the input size has a statistically significant impact on the correctness of participants' responses ($p = 0.016$). The effect of recursion type on correctness is even highly statistically significant ($p < 0.001$). The interaction between input size and recursion type is also statistically significant ($p = 0.045$). Those results indicate that the effect of Input Size on correctness depends on the recursion type of the task and vice versa. Table 4.8 provides the detailed ANOVA results for correctness.

- **Response Time** The effect of input size on response time is highly statistically significant ($p = 0.001$). This indicates that the size of the input has a significant impact on

how long it takes participants to complete the task. Furthermore, the different recursion types significantly impact response times (p = 0.027). However, the interaction between input size and recursion type does not reach statistical significance for the response time (p = 0.133), suggesting that the effect of input size on response time is consistent across different recursion types. Table 4.9 outlines the comprehensive results for response time.

As a summary, recursion type solely significantly influences the impact of input size on response time, without affecting correctness.

Table 4.8: Two-way ANOVA Results for Correctness

Source	Sum of Squares	df	F	P-value
Input Size	2.0	3	3.6	**0.016**
Recursion Type	4.8	2	12.9	**< 0.001**
Input Size × Recursion Type	2.5	6	2.2	**0.045**
Residual	29.3	156	—	—

Table 4.9: Two-way ANOVA Results for Response Time

Source	Sum of Squares	df	F	P-value
Input Size	58721.3	3	6.2	**< 0.001**
Recursion Type	23090.4	2	3.7	**0.027**
Input Size × Recursion Type	31323.8	6	1.7	0.133
Residual	489177.6	156	—	—

Tukey HSD To further understand which specific groups differ significantly in terms of correctness and response time, we rely on the Tukey HSD as *post-hoc* test. To determine practical significance, we performed the Cliff's Delta test to calculate effect sizes for every group that showed statistical significance in the Tukey HSD results.

- **Correctness** We primarily observed significant differences in correctness between tasks involving tree recursive functions with input sizes 3 and 4 compared to functions with other types of recursion and input sizes. These differences were characterized by large effect sizes, indicating substantial practical significance. Table 4.10 lists the results for every two groups that led to significantly different results with their corresponding effect size. The Appendix lists the Tukey HSD results for all combinations (see Table A.3).

- **Response Time** The Tukey HSD results for response time reflect a similar pattern to those observed in correctness. However, the number of significant group differences is notably smaller for response time. Tasks involving tree recursive with input sizes 3

and 4 resulted in significantly longer response times compared to other groups, with large effect sizes, indicating that these conditions were notably more time-consuming for participants. Furthermore, within the same input size, tree recursive functions led to significantly longer response times in contrast to linear recursive functions. Table 4.11 lists the results for every two groups that led to significantly different results with their corresponding effect size. The Appendix lists the Tukey HSD results for all combinations (see Table A.4).

Table 4.10: Significant Tukey HSD Results for Correctness with Effect Sizes

Group 1	Group 2	Mean Difference	P-value	Lower	Upper	Effect Size (Cliff's Delta)
1_Linear	3_Tree	-0.6667	**0.0024**	-1.1915	-0.1418	0.67 (large)
1_Linear	4_Tree	-0.5333	**0.0427**	-1.0582	-0.0085	0.53 (large)
1_Tail	3_Tree	-0.7000	**0.0028**	-1.2567	-0.1433	0.70 (large)
1_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100	0.57 (large)
1_Tree	3_Tree	-0.6667	**0.0024**	-1.1915	-0.1418	0.67 (large)
1_Tree	4_Tree	-0.5333	**0.0427**	-1.0582	-0.0085	0.53 (large)
2_Linear	3_Tree	-0.7333	**0.0005**	-1.2582	-0.2085	0.73 (large)
2_Linear	4_Tree	-0.6000	**0.0111**	-1.1248	-0.0752	0.60 (large)
3_Linear	3_Tree	-0.5333	**0.0427**	-1.0582	-0.0085	0.53 (large)
3_Tail	3_Tree	-0.7000	**0.0028**	-1.2567	-0.1433	0.70 (large)
3_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100	0.57 (large)
3_Tree	4_Linear	0.5333	**0.0427**	0.0085	1.0582	-0.53 (large)
3_Tree	4_Tail	0.7000	**0.0028**	0.1433	1.2567	-0.70 (large)
4_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100	0.57 (large)

Table 4.11: Significant Tukey HSD Results for Response Time with Effect Sizes

Group 1	Group 2	Mean Difference	P-value	Lower	Upper	Effect Size (Cliff's Delta)
1_Linear	3_Tree	74.7007	**0.0177**	6.8484	142.5530	-0.62 (large)
1_Tree	3_Tree	76.0697	**0.0142**	8.2174	143.9220	-0.62 (large)
1_Tree	4_Tree	68.3094	**0.0468**	0.4571	136.1617	-0.52 (large)
3_Linear	3_Tree	69.3456	**0.0403**	1.4933	137.1979	-0.56 (large)

4.3.3 Discussion

Our analysis demonstrates a significant impact of recursion type and input size on the correctness and response times of novice programmers when comprehending recursive functions. The data indicates that tree recursion, in particular, poses substantial challenges compared to linear and tail recursion, especially with increasing input size.

Correctness Correctness in tasks with tree recursive functions declines sharply with larger input sizes. The two-way ANOVA test indicates that input size and the type of recursion independently and their interaction have a highly significant effect on response time. The Tukey HSD test identifies significant differences primarily between tree-recursive tasks with inputs 3 and 4 and other recursion types, all with large effect sizes. These results suggest that the complexity inherent in tree recursion significantly hampers the ability of novice programmers to comprehend and solve these problems accurately.

Response Time The response time for tasks with tree recursive methods also increases markedly with input size. The two-way ANOVA test indicates that input size and the type of recursion independently have a highly significant effect on response time. However, the interaction between input size and type of recursion is not statistically significant, suggesting that the influence of input size on response time is consistent across recursion types. The Tukey HSD test identifies significant differences in response times between tree recursion tasks at higher input sizes and other groups with large effect sizes, indicating that tree recursion substantially increases the time required to complete tasks.

Implications The overall effect of input size on correctness and response time while comprehending recursive functions is discussed in Section 4.1.3. Our third research question specifically examines the *interaction effect* between recursion type and input size.

The significant interaction between recursion type and input size on correctness highlights the combined challenges when dealing with more complex recursion types and larger input sizes. Specifically, tree recursion with larger inputs poses substantial difficulties, likely due to the increased cognitive load required to process multiple recursive paths and deeper recursion levels. This result aligns with the cognitive load theory, suggesting that the simultaneous increase in recursion complexity and input size can overwhelm the working memory capacity of participants (see Section 2.2). This theory is supported by our post-questionnaire. Several participants noted in the post-questionnaire that they encountered the greatest difficulties with comprehending tree recursive functions, especially with higher input sizes. They reported the need to remember two values simultaneously within the same recursive call and the challenge of preserving the correct order of calls as primary reasons for this difficulty. Furthermore, some participants described comprehending tail recursive functions as simpler than linear recursive functions that were not tail recursive. However, while our statistical analysis supports the heightened difficulty associated with tree recursion, it does not confirm that tail recursion is inherently simpler than non-tail linear recursion.

Contrary to the plausible assumption that an input size of 4 would impose the highest challenge, the significant results from the Tukey HSD analysis for both correctness and

response time predominantly involve groups with input size 3. One possible explanation for this observation emerges from the post-questionnaire responses. Several participants reported that as the tasks became more complex, they changed their problem-solving strategies. Specifically, when faced with an input size of 4, which involved the highest number of recursive calls, they described shifting from step-by-step tracing each line of code to trying to grasp the general functionality of the recursive function and calculating the output directly. This possible strategic adjustment may have reduced the cognitive load associated with input size 4, as participants avoided the exhaustive mental effort of line-by-line tracing.

Hence, tasks with input size 3 may have represented a threshold where the complexity was high enough to strain working memory but not sufficient to prompt a change in strategy. Participants continued to trace the recursive calls line by line, leading to increased cognitive load and, consequently, more significant impacts on accuracy and response time. This suggests that the heightened cognitive demands of comprehending and processing the recursive calls at input size 3 were more taxing than the alternative strategies employed at input size 4. This theory presents an interesting opportunity for further research and exploration. As previously discussed in Section 4.1.3, the discrepancies observed between correctness and response time may be attributed to the speed-accuracy tradeoff. This phenomenon suggests that participants might prioritize accuracy over speed or vice versa when faced with complex tasks. The strategic adjustments reported by participants—such as shifting from step-by-step tracing to grasping the general functionality—could influence this tradeoff, leading to variations in correctness and response time that do not perfectly align.

For educators, these results emphasize the need to adopt targeted instructional strategies when teaching tree recursion. Introducing complex recursion types like tree recursion should be paired with smaller input sizes initially to manage cognitive load effectively. The thesis suggests that increasing input sizes as participants become more comfortable with the recursion type could facilitate comprehension. Additionally, understanding these challenges can inform the design of programming curricula that improve the preparation of novice programmers to handle complex recursive functions.

From a research perspective, when designing studies on program comprehension with specific input size, as multiple researchers did before (e.g. [1, 20, 28]), acknowledging the interaction effect of input size and type of recursion is crucial. Researchers should consider both recursion type and input size as interrelated factors that jointly influence learning outcomes. Future research on program comprehension with recursive functions that use an unspecific input size n (as several studies did before, e.g. [23, 46]), could improve control over their results and limit ambiguous interpretations not only by choosing a specific and sensible input size but also by taking the type of recursion into account. This awareness can lead to more nuanced experimental designs and more accurate interpretations of data, ultimately contributing to the development of more effective educational strategies and tools for comprehending recursion.

4.4 Effect of Type of Recursion on the Effect of Input Size on Visual Attention

This section explores the results obtained from the *eye-tracking data* collected throughout the experiment, examining the impact of recursion type on the relationship between input size and visual attention of novice programmers in understanding recursive functions, thus addressing our *fourth research question*.

4.4.1 Descriptive Statistics

The *descriptive statistics* provide an overview of general trends in our eye-tracking data grouped by the task input size and recursion type, providing initial insight into our fourth research question. We analyzed eye-tracking data from an average of 15 answers for each combination of recursion type and input size, totaling 180 records. Table 4.12 shows the data we gathered.

Table 4.12: Eye-Tracking Metrics by Recursion Type and Input Size

Recursion Type	Input	Mean Fixation Count ± SD	Mean Saccade Count ± SD	Mean Fixation Duration (s) ± SD	Mean Fixation Count/s ± SD
Linear	1	151 ± 52	151 ± 52	0.207 ± 0.02	3.63 ± 0.37
	2	158 ± 63	157 ± 63	0.217 ± 0.05	3.46 ± 0.48
	3	159 ± 70	159 ± 70	0.226 ± 0.04	3.38 ± 0.44
	4	161 ± 51	161 ± 51	0.261 ± 0.07	3.05 ± 0.62
Tail	1	261 ± 104	261 ± 104	0.200 ± 0.03	3.81 ± 0.41
	2	239 ± 110	239 ± 110	0.230 ± 0.07	3.55 ± 0.84
	3	228 ± 111	228 ± 111	0.243 ± 0.11	3.58 ± 0.95
	4	366 ± 165	366 ± 165	0.229 ± 0.05	3.31 ± 0.60
Tree	1	149 ± 55	149 ± 55	0.209 ± 0.06	3.85 ± 0.99
	2	263 ± 146	263 ± 146	0.203 ± 0.03	3.81 ± 0.49
	3	403 ± 193	403 ± 193	0.224 ± 0.05	3.47 ± 0.47
	4	367 ± 277	367 ± 277	0.223 ± 0.05	3.48 ± 0.53

Mean Fixation Count

- **Linear Recursion:** The mean fixation count for linear recursive functions shows a modest increase as the input size grows. Starting at 151 fixations for input size 1, it slightly rises to 158 for input size 2, 159 for input size 3, and reaches 161 for input size 4. This gradual increase suggests that as the complexity of linear recursion tasks

increases, participants require more fixations to process the information, indicating a moderate escalation in visual attention and cognitive effort.

- **Linear Recursion - Tail Recursion:** Tail recursive functions exhibit a varied pattern in mean fixation count. Beginning with a high of 261 fixations for input size 1, the count decreases to 239 for input size 2 and further to 228 for input size 3. However, there is a substantial jump to 366 fixations at input size 4. The sharp increase at input size 4 might suggest a significant rise in visual processing demands due to increased task complexity.
- **Tree Recursion:** Tree recursive functions show the most pronounced increase in mean fixation count with task complexity. Starting at 149 fixations for input size 1, it nearly doubles to 263 for input size 2 and peaks at 403 for input size 3. Although there is a slight decrease to 367 fixations at input size 4, the overall trend indicates that tree recursion tasks become substantially more demanding visually as the input size increases, reflecting heightened cognitive load and processing requirements.

Figure 4.9 visualizes the progressions for the mean fixation count for each input size and type of recursion.

Mean Fixation Duration

- **Linear Recursion:** The mean fixation duration in linear recursive functions increases steadily with their input size. It starts at 0.207 seconds for input size 1 and rises to 0.217 seconds for input size 2, 0.226 seconds for input size 3, reaching 0.261 seconds for input size 4. This upward trend suggests that participants spend more time per fixation as task complexity grows, likely due to the need for deeper processing and information integration in more complex tasks.
- **Linear Recursion - Tail Recursion:** Tail recursive functions display slight fluctuations in mean fixation duration without a clear linear trend. Starting at 0.200 seconds for input size 1, it increases to 0.230 seconds for input size 2 and peaks at 0.243 seconds for input size 3, then slightly decreases to 0.229 seconds for input size 4. This pattern indicates that while there is some variation, the time spent per fixation remains relatively consistent. Those results suggest that participants maintain a stable processing rate per fixation across different task complexities.
- **Tree Recursion:** the mean fixation duration remains relatively stable regarding tree recursive functions. It begins at 0.209 seconds for input size 1, slightly decreases to 0.203 seconds for input size 2, and then increases to 0.224 seconds and 0.223 seconds for input sizes 3 and 4, respectively. The minimal changes imply that despite the increased number of fixations, the time spent on each fixation does not vary substantially, possibly reflecting consistent processing strategies even as tasks become more complex.

Figure 4.10 visualizes the progressions for the mean fixation duration for each input size and recursion type.

Mean Fixation Count Per Second

- **Linear Recursion:** The mean fixations per second for linear recursive functions show a decreasing trend with increasing input size. Starting at 3.631 fixations per second for input size 1, it decreases to 3.461 for input size 2, 3.384 for input size 3, and further to 3.046 for input size 4. This decline suggests that participants slow their visual scanning rate as the task becomes more complex, potentially due to increased cognitive processing demands.
- **Linear Recursion - Tail Recursion:** Tail recursive functions exhibit a slight decrease in mean fixations per second with increasing input size. Beginning at 3.808 for input size 1, it decreases to 3.554 for input size 2, remains relatively stable at 3.578 for input size 3, and drops to 3.305 for input size 4. The relatively minor changes indicate that participants maintain a reasonably consistent visual scanning rate, even as task complexity increases.
- **Tree Recursion:** For tree recursive functions, the mean fixations per second decrease from 3.849 at input size 1 to 3.812 for input size 2 and further to 3.467 for input size 3, with a slight increase to 3.475 at input size 4. This overall decrease suggests that as the complexity of tree recursion tasks grows, participants reduce their fixation rate, likely due to the higher cognitive load requiring more deliberate processing.

Figure 4.11 visualizes the progressions for the mean fixations per second for each input size and type of recursion.

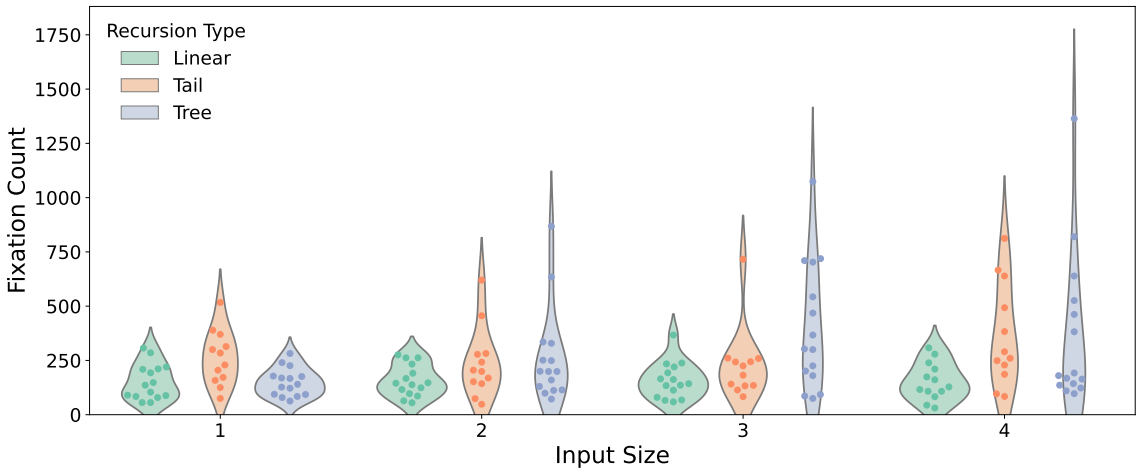


Figure 4.9: Fixation Count for each Input Size and Recursion Type

4.4.2 Statistical Tests

Two-Way ANOVA The two-way ANOVA test can show us whether there is a *significant interaction* between input size and the eye-tracking metrics we calculated.

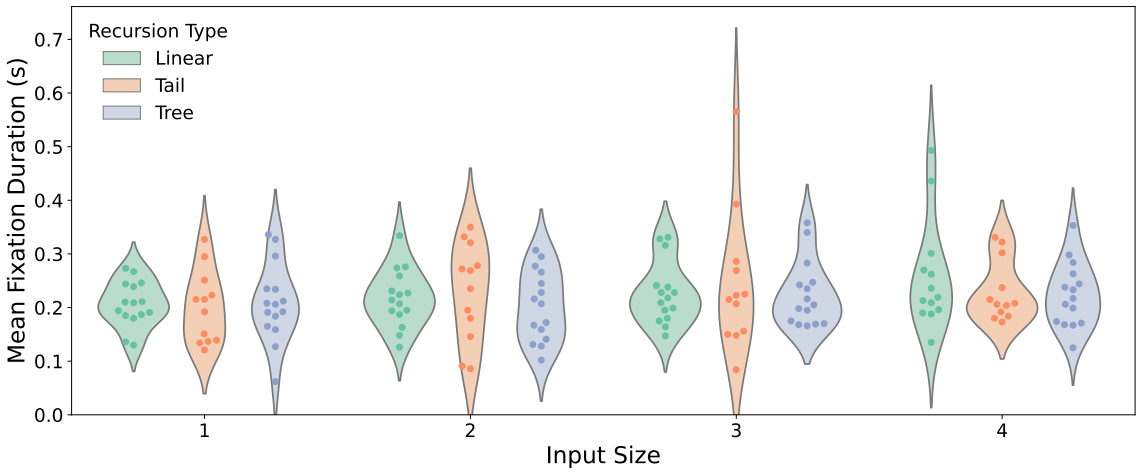


Figure 4.10: Mean Fixation Duration for each Input Size and Recursion Type

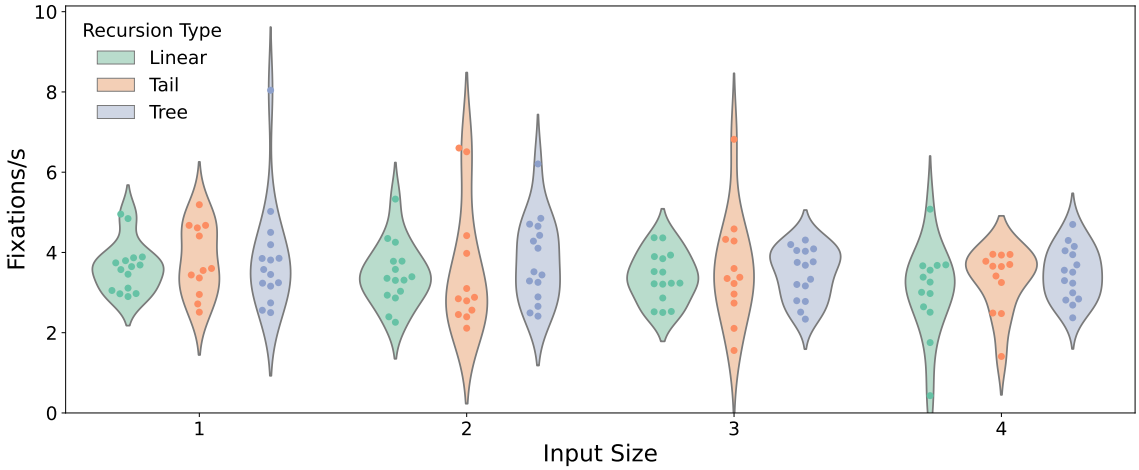


Figure 4.11: Fixation Count per Second for each Input Size and Recursion Type

- **Fixation Count** The mean number of fixations differs significantly across the different input sizes ($p = 0.04$). Furthermore, the effect of recursion type on fixations is highly statistically significant ($p < 0.01$). The interaction between input size and recursion type is close to significance ($p = 0.06$) but still > 0.05. Those results suggest a trend where the effect of input size on fixations may depend on the recursion type, but the evidence is not strong enough to conclusively state that an interaction exists. In conclusion, both input size and recursion type independently influence fixation counts, but the interaction between those variables is insignificant. Table 4.13 presents the detailed ANOVA results for fixation count.

- **Fixation Duration** The p-value of 0.216 is greater than 0.05, indicating that the effect of the input size of our recursive functions on mean fixation duration is not statistically significant. This suggests that the different input sizes do not lead to significant differences in the mean duration of participants' fixations. With a p-value of 0.630, the

effect of recursion type is also not statistically significant, so the type of recursion does not significantly impact the mean fixation duration. The interaction between input size and recursion type is not statistically significant ($p = 0.887$). This indicates that the effect of input size on mean fixation duration does not depend on the recursion type and vice versa. Since fixation duration is often associated with cognitive processing load, the lack of significant differences implies that the cognitive effort per fixation does not vary substantially with different tasks or recursion types in this study. Table 4.14 presents the detailed ANOVA results for fixation duration.

- **Fixation Count Per Second** The p-value for the effect of the input size on the functions is greater than 0.05 (0.152), indicating that the impact of input size on fixations per second is not statistically significant. With a p-value of 0.331, the effect of recursion type on the fixations per second is also not statistically significant. Furthermore, the interaction between input size and recursion type has no statistically significant impact on the fixations per second ($p = 0.993$). Those results suggest that the visual scanning rate of participants is consistent across tasks and recursion complexities. The results imply that, in this context, fixations per second may not be sensitive to changes in input size or recursion complexity, and other eye-tracking measures may be more informative in understanding the cognitive processing of participants during these tasks. Table 4.15 presents the detailed ANOVA results for fixations per second.

Overall, the type of recursion only significantly affects the fixation count. It is crucial to notice that the number of fixations correlates with the response time, which we discussed in Section 4.3. Since we found no significant interaction effect between input size and recursion type, we can conclude that the effect of recursion type does not significantly affect the effect of input size on our eye-tracking metrics.

Table 4.13: Two-Way ANOVA Results for Fixation Count

Source	Sum of Squares	df	F	P-value
Input Size	299991	3	2.846	**0.040**
Recursion Type	639984	2	9.106	**< 0.001**
Input Size × Recursion Type	429642	6	2.038	0.064
Residual	5411800	154	—	—

Table 4.14: Two-Way ANOVA Results for Mean Fixation Duration

Source	Sum of Squares	df	F	P-value
Task Type	0.024	3	1.501	0.216
Recursion Type	0.005	2	0.463	0.630
Task Type × Recursion Type	0.012	6	0.386	0.887
Residual	0.829	154	—	—

Table 4.15: Two-Way ANOVA Results for Fixation Count Per Second

Source	Sum of Squares	df	F	P-value
Input Size	5.189	3	1.786	0.152
Recursion Type	2.157	2	1.113	0.331
Input Size × Recursion Type	0.745	6	0.128	0.993
Residual	149.151	154	—	—

Tukey HSD For the completeness of our results, we performed the Tukey HSD test as *post-hoc* test to understand which specific group means differ significantly in terms of *fixation count*. To determine practical significance, we performed the Cliff's Delta test to calculate effect sizes for every group that showed statistical significance in the Tukey HSD results.

The Tukey HSD test reveals that understanding tree recursive functions takes significantly more fixations than linear and tail recursive functions, especially at higher input sizes. This suggests that tree recursion tasks are more complex and require more visual attention. The significant differences mainly involve tree recursive functions with input size 3, highlighting it as the most demanding in terms of fixation count. Notably, the results show that tree recursion at input size 3 is even more demanding in terms of fixation count than linear recursion at input size 4. Large effect sizes characterized these differences, indicating substantial practical significance. Table 4.16 lists the results for every two groups that led to significantly different results with their corresponding effect sizes. The Appendix contains the Tukey HSD test results of all combinations (see Table A.5).

Table 4.16: Significant Tukey HSD Results for Fixation Count with Effect Sizes

Group 1	Group 2	Mean Difference	P-value	Lower	Upper	Effect Size (Cliff's Delta)
1_Linear	3_Tree	252.07	**0.0162**	24.88	479.26	**-0.57 (large)**
1_Tree	3_Tree	253.80	**0.0149**	26.61	480.99	**-0.59 (large)**
2_Linear	3_Tree	245.60	**0.0220**	18.41	472.79	**-0.55 (large)**
3_Linear	3_Tree	244.60	**0.0230**	17.41	471.79	**-0.57 (large)**
3_Tree	4_Linear	-250.13	**0.0271**	-485.90	-14.37	**0.55 (large)**

4.4.3 Discussion

Our analysis reveals that the type of recursion significantly impacts the visual attention of participants when comprehending recursive functions, as evidenced by differences in fixation counts across recursion types and input sizes. Specifically, tree recursive functions demand substantially more visual attention, indicated by a higher mean fixation count,

compared to linear and tail recursive functions. This effect becomes more pronounced with larger input sizes.

Fixation Count The fixation count serves as a measurement for the amount of visual attention and cognitive effort required to process information [35]. The significant increase in fixation count for tasks with tree recursive functions, especially at input size 3, suggests that these tasks impose a higher cognitive load on participants. The two-way ANOVA confirmed that both input size and recursion type independently influence fixation counts. However, the interaction between input size and recursion type was not statistically significant, indicating that the effect of recursion type on fixation count might not significantly depend on input size within the tested range. Nevertheless, we must note that a small sample size, as in our study, diminishes the power of a statistical test, making it less likely to detect a true effect even if one exists.

The Tukey HSD post-hoc test revealed that significant differences in fixation count mainly occur between tasks with tree recursive functions at larger input sizes and those involving other types of recursion. The significant effect sizes underscore the practical significance of these results. This indicates that tree recursion is inherently more complex, possibly due to its non-linear structure and the exponential growth of recursive calls with input size.

The observed increase in fixation count for tasks with tree recursive functions can be interpreted through the lens of the *cognitive load theory* (see 2.2). Tree recursion, with its branching structure and multiple recursive calls, imposes a higher *cognitive load* due to the complexity inherent in the task. As the input size increases, the cognitive load intensifies, requiring participants to hold and process more information simultaneously in working memory. The increased fixation count reflects the additional visual attention and cognitive resources needed to process these complex structures. This is supported by the post-questionnaire, in which participants reported that the comprehension of tree recursive functions posed the greatest challenge for them. In Section 4.3.3, we provide further interpretation of this feedback.

Fixation Duration and Fixation Count Per Second Interestingly, mean fixation duration and fixations per second did not show significant differences across recursion types or input sizes. The two-way ANOVA results for fixation duration and fixations per second suggest that participants maintain a consistent processing rate per fixation regardless of task complexity. This stability may indicate that, while the number of fixations increases with task complexity, the cognitive effort per fixation does not vary significantly. Furthermore, the lack of a significant increase in fixation duration hints that even as the complexity of the tasks rises, participants' eyes continue to move around the code rather than fixating on a single point. This observation contrasts with reports from some participants in the post-questionnaire, where they mentioned fixating on a single point to think through the problem. Our eye-tracking data suggests that, despite their subjective experience of fixating, participants' visual attention remained active and distributed across different areas of the code. Determining which specific areas of the code are involved during these periods of cognitive processing remains an open question for future analysis.

Implications The overall effect of input size on eye movements while comprehending recursive functions is discussed in Section 4.2.3. Our fourth research question explicitly examines the interaction effect on gaze data between recursion type and input size. The results of our statistical analysis reveal that the interaction effect between these two groups is insignificant. Even if trends are present, they may not be large enough between our types of recursion to reach statistical significance. As explained in detail in Section 4.2.3, since our study involved a limited number of participants, the statistical power of our tests is reduced, making it less likely to detect a true effect even if one exists. Therefore, expanding the participant pool could enhance statistical power and allow for more thorough analyses of interaction effects.

Notably, similar to the significant Tukey HSD results observed in our third research question (see Tables 4.10 and 4.11), the significant differences in the Tukey HSD analysis for this research question also predominantly involve groups with input size 3 rather than the expected input size 4. This phenomenon was discussed earlier in Section 4.3.3, where we proposed that participants might change their problem-solving strategy when faced with the most complex tasks. Specifically, as the task reached a certain number of recursive calls (at input size 4 in our case), in the post-questionnaire, participants reported shifting from line-by-line code tracing to inferring the general functionality of the recursive function and calculating the output directly. This strategic shift may have reduced the cognitive load for input size 4 tasks. In contrast, tasks with input size 3 may have led to detailed tracing, resulting in a higher cognitive load reflected in the significant differences we observed.

In conclusion, while our results on the interaction effect of gaze data between recursion type and input size are not statistically significant, this does not imply that there is no meaningful answer to our research question. The insights from our post-questionnaire propose that cognitive load and eye movement patterns are influenced not just by task complexity but also by the problem-solving approaches employed by participants at different levels of difficulty.

5 Threats to Validity

While the study provides valuable insights into the effect of input size on the performance of students while comprehending recursive functions, it has *limitations* that are discussed in this section.

5.1 Internal Validity

Eye-tracking comes with its inevitable difficulties regarding threats to validity. During an experiment, the eye-tracking accuracy decreases gradually over time. This so-called *drift* indicates a deterioration of calibration [36]. It is caused by changes in the physiology of the eye, e.g., changes in wetness, or changing environmental conditions, e.g., light conditions. To diminish the effect of drift, we manually corrected the eye-tracking data by modifying the x and y values accordingly.

To increase the homogeneity of our code snippets, some of them are modified textbook examples. Thus, students might be familiar with the selected snippets, as many are commonly used in introductory computer science courses. This familiarity can result in top-down comprehension, as students may take shortcuts and provide answers without genuinely understanding the code. In the post-questionnaire, some students confirmed recognizing particular snippets and, in conclusion, taking shortcuts. Textbook example snippets pose another possible threat, as participants might expect its unaltered functionality, which means the modifications could be misleading. Nevertheless, as discussed in Section 3.5, using code from existing, similar studies and typical textbook examples makes the code comparable, student-friendly, and less prone to error [48].

The design introduces the possibility of individual differences affecting the results, as participants in each group may differ in ways unrelated to the experimental manipulation. For instance, variations in prior programming experience, familiarity with recursion, or cognitive abilities among participants could influence their performance and comprehension independent of the input size or recursion type. Those individual differences are inevitable, but we tried to minimize them by posing necessary requirements for the participation of a participant, e.g., being actively enrolled in the Bachelor of Computer Science or an equivalent program throughout the duration of the study. Additionally, we mitigated the threat by showing each participant code snippets with different input sizes and all types of recursion. The full requirements participants had to fulfill are listed in Section 3.4.3.

Furthermore, we define input size based on the depth of the recursion tree of the function rather than the length of the input data (see Section 3.2). While the depth of the recursion tree remains consistent across different tasks, the length of the input data varies between them.

This indicates that, although participants understood functions with the same recursion depths, they processed inputs of varying lengths. The variation in input length can impact the cognitive load and comprehension difficulty experienced by participants, independent of the recursion depth. Longer inputs may require more mental resources to interpret and may introduce additional complexity unrelated to the recursive structure itself. However, varying input lengths made it possible to choose a more diverse sample of snippets while keeping the number of recursive calls the same.

Another potential threat to the internal validity of our study arises from the difference in the number of recursive calls between linear (including tail recursion) and tree recursive functions, even when the recursion tree depth is consistent. This difference was inevitable in our design since, in our tree recursive functions, each function call generates exactly two further recursive calls, leading to an exponential increase in the total number of recursive calls. In contrast, linear recursion produces only one recursive call at each level. Consequently, the next smallest input size for a tree recursive function leads to more recursive calls than the next smallest size for a linear recursive function. While the depth of the recursion tree offers a valuable variable for comparing different types of recursion, this discrepancy also introduces a confounding variable that may independently influence the cognitive load and performance of participants: the number of recursive calls. The observed variations among linear and tree recursive functions may be attributed not only to the specific type of recursion employed but also to the varying number of recursive calls. Although this presents a challenge in evaluating linear versus tree recursion, it does not affect research questions one and two, nor does it impede the assessment of how input size influences the understanding of functions of a single type of recursion.

5.2 External Validity

A crucial point is that those simple code snippets do not represent real-world programs. The snippets are recursive, short, and simple, using numbers exclusively as input to support the analysis of our research questions specifically. Therefore, transferring our results to more complex snippets with different input types or iterative code should be considered carefully.

Further, our findings can not be easily compared to other hardware and software tools since there may be significant differences in their results for the same data set [9].

Moreover, some previous studies show that students tend to have a harder time understanding recursive code compared to iterative code [16, 21]. According to a study by Núñez-Varela et al., students ranked recursion as the hardest code category among conditional, instruction sequence, or iterative control structure categories [26]. Consequently, as our sample predominantly consists of novice programmers, introducing a selection bias that limits the generalizability of our results to experienced programmers, who might process recursive code differently than their less experienced counterparts.

Several participants mentioned in the post-questionnaire that one of the main challenges they faced during the task was remembering the output. This difficulty may have contributed to an increase in cognitive load as the size of the input grew. However, outside of a research setting, it is generally feasible to take notes while engaging in program comprehension. Allowing participants to write things down could produce different results, as they would

not be solely dependent on their memory for calculations. However, we chose not to permit note-taking to prevent distractions that might divert their gaze from the screen, which could potentially alter the eye-tracking data.

5.3 Construct Validity

A potential threat to construct validity in this study is the limited range of input sizes evaluated due to the constrained time frame of conducting a bachelor's thesis. This narrow spectrum may not have been sufficient to capture significant impacts of increasing input sizes on certain measurements, such as fixation count per second, mean fixation duration, or response times. It is plausible that more substantial effects on cognitive load and comprehension would emerge with larger input sizes, which could more intensely challenge the participants' mental processing capabilities.

Furthermore, other eye-tracking metrics, which were not included in our analysis due to time constraints, might reveal significant differences when comparing input sizes. While we focused on simple metrics such as fixation count, mean fixation duration, and fixation count per second, there are additional eye-tracking measures – such as pupil dilation or scanpath patterns – , that could provide further insights into cognitive load and processing strategies. Thus, the study does not fully capture the complexity of how input size affects program comprehension, potentially overlooking significant effects that other eye-tracking metrics might reveal.

5.4 Conclusion Validity

Furthermore, a striking feature in our evaluation is the lack of significant differences in correctness and eye-tracking metrics between specific input sizes. However, even though our results for certain metrics are not statistically relevant, does not mean that there is no relevant answer regarding the effect of input size on the comprehension of recursive functions on those metrics overall. Several threats can lead to these results.

One plausible explanation for this is that the sample size may not have been sufficient to detect more subtle effects. As explained in Section 4.2.3, a small sample size, as in our study, diminishes the power of a statistical test, making it less likely to detect a true effect even if one exists.

Another threat is that the code snippets might have been too simple to elicit measurable differences in correctness and certain eye-tracking metrics. The directness of the snippets may not have adequately challenged participants' cognitive processing abilities, potentially hiding differences that could arise with more complex or demanding code.

6 Concluding Remarks

6.1 Conclusion

To close the research gap on how different input sizes affect the ability of novice programmers to comprehend recursive functions, we conducted a controlled lab experiment with twelve computer science undergraduates with similar programming experience. Each participant was asked to determine the output of 15 Java code snippets varying in recursion type and input size, evaluated across four input sizes. We recorded their response times and accuracy and used an eye-tracking device to assess visual attention and cognitive load. In the following subsections, we summarize the results of each research question.

6.1.1 Research Question 1 & 2

> ***RQ1:*** *How does the input size affect students' response times and correctness when understanding recursive functions?*
>
> ***RQ2:*** *How does input size affect students' visual attention when understanding recursive functions?*

Our study found that increasing the input size in recursive functions significantly affected novice programmers' comprehension. As input sizes grew, participants had significantly longer response times and significantly reduced correctness. Although eye-tracking data showed trends of increased fixation counts and durations with larger inputs, these changes were not statistically significant. The results support the cognitive load theory, which states that processing more information increases learners' cognitive load. In recursive functions, larger inputs lead to greater complexity, placing higher demands on working memory. Consequently, participants may have longer response times and a greater likelihood of errors.

Notably, there is a significant jump in difficulty between input sizes 3 and 4, indicating that recursion complexity may exponentially increase beyond a certain threshold, overwhelming cognitive capacities. Participants may have changed their problem-solving strategies, moving from detailed line-by-line tracing to a more holistic understanding. This shift could explain the variations in response times and correctness.

Educators should consider this jump and possible change of strategies when designing instructional materials and exercises, ensuring that input sizes are scaled appropriately to match the developing comprehension skills of novice programmers. For researchers,

this emphasizes the importance of carefully selecting input sizes in experimental designs. Multiple studies use an unspecific input n for their recursive code snippets [23] [46]. This unspecific input size can introduce undue complexity, which may confound results or mask the effects of other variables under investigation. By carefully selecting and standardizing input sizes in recursive functions, researchers can improve the validity and reliability of their studies. Furthermore, this control enhances the comparability of results across different experiments.

6.1.2 Research Question 3 & 4

> ***RQ1:*** *How does the input size affect students' response times and correctness when understanding recursive functions?*
>
> ***RQ2:*** *How does input size affect students' visual attention when understanding recursive functions?*

Our study further explored how the type of recursion (linear, tail, and tree) interacts with input size to affect novice programmers' correctness, response time, and visual attention during program comprehension. We found that tree recursion significantly challenges participants more than linear and tail recursion, especially at higher input sizes. Notably, the interaction effect between input size and type of recursion was significant for correctness, indicating that the combined impact of recursion type and input size specifically affects novice programmers' accuracy in understanding recursive functions. Significant differences were particularly evident in tasks involving input sizes of 3 and 4, indicating a potential shift in problem-solving strategies as complexity increases. Participants may shift from detailed code tracing to attempting to understand the overall function behavior, which could affect their performance.

For educators, our results highlight the importance of carefully introducing recursion types in programming curricula. Since tree recursion poses significantly greater challenges, it may be beneficial to first build a strong foundation with linear and tail recursion before introducing tree recursion. For researchers, our results emphasize the need to consider both recursion type and input size as critical variables in program comprehension studies. We provide an opportunity to improve the comparability of future studies using recursive code by supporting the determination of a suitable input size with a fitting type of recursion for their applied functions by controlling the difficulty of tasks. Furthermore, understanding the thresholds at which novice programmers encounter challenges and might change their problem-solving strategies can improve the design of experiments and educational interventions.

6.2 Future Work

While our study provides valuable insights into how input size and recursion type affect novice programmers' comprehension of recursive functions, several avenues for future research remain to deepen our understanding and address the limitations identified.

6.2.1 Further Analysis of Gathered Data

The restricted time frame of a bachelor's thesis also imposed limitations on our analysis. Beyond increasing the sample size, a deeper analysis of the existing data could yield additional insights. Specifically, by examining fourth-order eye-tracking data (see 2.6), such as scan paths and linearity metrics, we can explore the sequential patterns of visual attention during code comprehension. By defining areas of interest within the code snippets, we can quantify how participants navigated between different code elements and whether certain recursion types or input sizes led to more erratic or systematic viewing patterns.

Additionally, the reports generated from the post-questionnaire provide valuable insights that may add to the scope of our analysis. Participants indicated that they jumped more frequently between code lines as the number of recursive calls increased. Some participants also mentioned fixating on a single point while thinking through the problem. By examining the scan paths and fixation distributions, we can corroborate these self-reported strategies with empirical eye-tracking data, providing a more comprehensive understanding of the cognitive processes involved.

6.2.2 Replication and Future Studies

This subsection outlines key areas for replication and suggests directions for future studies that could enhance the robustness and applicability of our results. It explores additional dimensions that influence the comprehension of recursive functions.

Larger Sample Size One immediate direction for future work is to replicate this study with a larger participant pool. As noted in our evaluation (see Chapter 4), the limited number of participants reduces the statistical power of our analyses, making it challenging to detect smaller effect sizes and potentially obscuring true effects. A larger sample size would enhance the robustness of the statistical tests, increase the generalizability of the results, and allow for more detailed subgroup analyses. This expansion could provide more conclusive evidence regarding the effect of input size on both behavioral and eye-tracking metrics, thereby strengthening the validity of our conclusions.

Exploration with More Complex Code Snippets Another avenue for future research involves extending the study to include more complex code snippets. The current study focused on relatively straightforward recursive functions to isolate the effects of input size and recursion type. Incorporating more complex programming constructs, such as nested recursion, mutual recursion, or tasks with additional control structures (e.g., loops and conditional statements), could provide insights into how participants handle increased cognitive demands. This exploration could help determine whether the observed effects persist with greater code complexity and how instructional strategies might need to adapt accordingly.

Incorporating Individual Differences Future research should also consider individual differences among learners, such as prior programming experience, cognitive abilities,

and learning styles. These factors may influence how participants interact with recursive functions and respond to varying input sizes and recursion types. By collecting data on these variables, we can perform more sophisticated analyses to identify patterns and tailor instructional approaches to meet diverse learner needs.

In summary, future work should focus on expanding the participant base, delving deeper into the collected data, exploring more complex programming tasks, and considering individual learner differences. These efforts will enhance our understanding of recursion comprehension and inform the development of effective teaching strategies to address the challenges identified in this study.

A

Appendix: Data

A.1 Descriptive Statistics

Table A.1: Summary of Task Performance Based on Recursion Type, Correctness, and Response Times

Task	Recursion Type	Correctness (%)	Mean Time (s)	Mean Correct Only Time (s)
A1	Tree	3/3 (100)	38.7 (± 12.8)	38.7 (± 12.8)
A2	Tree	3/3 (100)	53.2 (± 15.5)	53.2 (± 15.5)
A3	Tree	1/3 (33.3)	179.0 (± 74.8)	36.0 (± 62.3)
A4	Tree	1/3 (33.3)	153.3 (± 121.6)	37.1 (± 64.3)
B1	Linear	3/3 (100)	25.1 (± 3.6)	25.1 (± 3.6)
B2	Linear	3/3 (100)	25.6 (± 6.8)	25.6 (± 6.8)
B3	Linear	3/3 (100)	25.9 (± 8.1)	25.9 (± 8.1)
B4	Linear	2/3 (66.7)	135.8 (± 152.2)	32.9 (± 39.1)
C1	Linear	2/3 (66.7)	59.9 (± 14.9)	44.3 (± 39.6)
C2	Linear	2/3 (66.7)	91.0 (± 10.1)	58.8 (± 51.7)
C3	Linear	0/3 (0)	55.0 (± 29.7)	0.0 (± 0.0)
C4	Linear	1/3 (33.3)	59.5 (± 12.2)	24.0 (± 41.5)
D1	Tail	1/3 (33.3)	98.4 (± 10.6)	36.9 (± 63.8)
D2	Tail	1/3 (33.3)	96.7 (± 44.6)	32.9 (± 56.9)
D3	Tail	2/3 (66.7)	76.4 (± 26.0)	53.1 (± 52.5)
D4	Tail	2/3 (66.7)	146.1 (± 55.6)	113.8 (± 104.8)
G1	Tail	3/3 (100)	32.9 (± 10.0)	32.9 (± 10.0)
G2	Tail	3/3 (100)	23.9 (± 1.6)	23.9 (± 1.6)
G3	Tail	3/3 (100)	36.3 (± 9.8)	36.3 (± 9.8)
G4	Tail	3/3 (100)	58.6 (± 21.7)	58.6 (± 21.7)

Continued on next page

Table A.1: (Continued) Summary of Task Performance Based on Recursion Type, Correctness, and Response Times

Task	Recursion Type	Correctness (%)	Mean Time (s)	Mean Correct Only Time (s)
H1	Tail	3/3 (100)	55.3 (± 14.2)	55.3 (± 14.2)
H2	Tail	2/3 (66.7)	56.0 (± 32.9)	47.5 (± 45.6)
H3	Tail	3/3 (100)	60.2 (± 7.4)	60.2 (± 7.4)
H4	Tail	3/3 (100)	105.6 (± 54.9)	105.6 (± 54.9)
I1	Linear	3/3 (100)	25.2 (± 17.9)	25.2 (± 17.9)
I2	Linear	3/3 (100)	36.5 (± 2.7)	36.5 (± 2.7)
I3	Linear	3/3 (100)	40.6 (± 10.9)	40.6 (± 10.9)
I4	Linear	3/3 (100)	38.3 (± 5.9)	38.3 (± 5.9)
J1	Tree	2/3 (66.7)	30.8 (± 7.4)	23.1 (± 20.2)
J2	Tree	3/3 (100)	29.7 (± 2.2)	29.7 (± 2.2)
J3	Tree	1/3 (33.3)	35.2 (± 23.9)	6.7 (± 11.5)
J4	Tree	2/3 (66.7)	36.6 (± 6.0)	23.6 (± 21.2)
L1	Tree	1/3 (33.3)	71.2 (± 18.2)	24.4 (± 42.3)
L2	Tree	0/3 (0)	109.2 (± 77.0)	0.0 (± 0.0)
L3	Tree	0/3 (0)	73.4 (± 18.7)	0.0 (± 0.0)
L4	Tree	1/3 (33.3)	251.3 (± 46.6)	75.7 (± 131.1)
N1	Tree	3/3 (100)	27.9 (± 11.4)	27.9 (± 11.4)
N2	Tree	0/3 (0)	48.4 (± 25.4)	0.0 (± 0.0)
N3	Tree	0/3 (0)	133.4 (± 138.2)	0.0 (± 0.0)
N4	Tree	0/3 (0)	43.2 (± 18.4)	0.0 (± 0.0)
O1	Linear	2/3 (66.7)	43.0 (± 16.9)	22.7 (± 20.7)
O2	Linear	3/3 (100)	37.7 (± 19.5)	37.7 (± 19.5)
O3	Linear	2/3 (66.7)	39.8 (± 15.3)	31.9 (± 28.3)
O4	Linear	3/3 (100)	34.5 (± 18.3)	34.5 (± 18.3)
Q1	Tail	3/3 (100)	89.6 (± 11.4)	89.6 (± 11.4)
Q2	Tail	2/3 (66.7)	117.3 (± 39.3)	63.9 (± 56.8)
Q3	Tail	2/3 (66.7)	71.5 (± 6.7)	47.1 (± 41.3)
Q4	Tail	2/3 (66.7)	105.8 (± 58.1)	48.3 (± 42.0)
R1	Linear	2/3 (66.7)	56.6 (± 27.5)	34.9 (± 40.2)
R2	Linear	2/3 (66.7)	57.3 (± 50.7)	48.3 (± 60.3)
R3	Linear	2/3 (66.7)	75.4 (± 26.3)	52.7 (± 52.3)

Continued on next page

Table A.1: (Continued) Summary of Task Performance Based on Recursion Type, Correctness, and Response Times

Task	Recursion Type	Correctness (%)	Mean Time (s)	Mean Correct Only Time (s)
R4	Linear	1/3 (33.3)	169.2 (± 117.6)	34.5 (± 59.7)
S1	Tree	3/3 (100)	34.4 (± 5.3)	34.4 (± 5.3)
S2	Tree	1/3 (33.3)	95.2 (± 36.1)	23.0 (± 39.9)
S3	Tree	0/3 (0)	162.3 (± 14.2)	0.0 (± 0.0)
S4	Tree	0/3 (0)	60.1 (± 29.8)	0.0 (± 0.0)

Table A.2: Summary of Eye-Tracking Metrics Based on Task and Recursion Type

Task	Recursion Type	Mean Fixation Count	Mean Fixation Duration	Mean Fixations/s
A1	Tree	149.0 (± 48.5)	0.188	3.86
A2	Tree	192.7 (± 59.8)	0.205	3.679
A3	Tree	574.0 (± 237.4)	0.231	3.222
A4	Tree	672.7 (± 613.7)	0.175	4.049
B1	Linear	87.3 (± 3.1)	0.212	3.517
B2	Linear	88.7 (± 26.1)	0.220	3.463
B3	Linear	96.3 (± 41.0)	0.192	3.710
B4	Linear	181.0 (± 138.6)	0.192	3.679
C1	Linear	210.0 (± 74.5)	0.221	3.443
C2	Linear	257.0 (± 21.9)	0.280	2.855
C3	Linear	166.3 (± 88.1)	0.255	3.090
C4	Linear	134.0 (± 92.5)	0.286	2.458
D1	Tail	353.3 (± 148.0)	0.225	3.517
D2	Tail	335.0 (± 251.7)	0.250	3.222
D3	Tail	374.3 (± 296.7)	0.174	4.465
D4	Tail	562.7 (± 222.8)	0.188	3.846
G1	Tail	119.3 (± 41.8)	0.208	3.648
G2	Tail	91.3 (± 54.1)	0.198	3.834
G3	Tail	119.0 (± 31.4)	0.237	3.329
G4	Tail	166.3 (± 74.5)	0.289	2.793
H1	Tail	249.0 (± 106.1)	0.161	4.414

Continued on next page

Table A.2: (Continued) Summary of Eye-Tracking Metrics Based on Task and Recursion Type

Task	Recursion Type	Mean Fixation Count	Mean Fixation Duration	Mean Fixations/s
H2	Tail	218.7 (± 57.7)	0.154	4.560
H3	Tail	248.7 (± 9.0)	0.168	4.171
H4	Tail	405.3 (± 226.2)	0.192	3.796
I1	Linear	110.7 (± 94.7)	0.170	4.258
I2	Linear	150.7 (± 14.8)	0.168	4.128
I3	Linear	146.7 (± 29.2)	0.201	3.671
I4	Linear	117.3 (± 10.1)	0.237	3.110
J1	Tree	88.0 (± 30.5)	0.289	2.826
J2	Tree	123.7 (± 67.5)	0.208	4.072
J3	Tree	123.0 (± 68.1)	0.203	3.667
J4	Tree	123.0 (± 13.0)	0.226	3.398
L1	Tree	225.7 (± 56.5)	0.251	3.218
L2	Tree	409.0 (± 398.0)	0.223	3.365
L3	Tree	298.3 (± 71.1)	0.176	4.091
L4	Tree	661.7 (± 148.3)	0.312	2.624
N1	Tree	106.0 (± 29.9)	0.185	3.935
N2	Tree	159.0 (± 79.7)	0.230	3.409
N3	Tree	446.7 (± 545.3)	0.301	2.844
N4	Tree	148.3 (± 44.9)	0.204	3.565
O1	Linear	154.3 (± 53.8)	0.198	3.633
O2	Linear	123.3 (± 68.0)	0.242	3.213
O3	Linear	109.7 (± 43.9)	0.282	2.745
O4	Linear	130.3 (± 99.4)	0.214	3.616
Q1	Tail	324.3 (± 56.6)	0.206	3.655
Q2	Tail	311.0 (± 131.6)	0.316	2.601
Q3	Tail	170.0 (± 79.5)	0.394	2.349
Q4	Tail	328.0 (± 280.0)	0.248	2.786
R1	Linear	193.0 (± 114.0)	0.233	3.303
R2	Linear	168.0 (± 84.9)	0.176	3.648
R3	Linear	273.7 (± 81.1)	0.202	3.703
R4	Linear	241.0 (± 94.8)	0.378	2.365
S1	Tree	178.0 (± 57.3)	0.132	5.408

Continued on next page

Table A.2: (Continued) Summary of Eye-Tracking Metrics Based on Task and Recursion Type

Task	Recursion Type	Mean Fixation Count	Mean Fixation Duration	Mean Fixations/s
S2	Tree	433.0 (± 175.0)	0.147	4.536
S3	Tree	573.7 (± 123.9)	0.211	3.509
S4	Tree	229.7 (± 132.3)	0.196	3.741

a.2 Statistical Tests

Table A.3: Tukey HSD Results for Correctness

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Linear	1_Tail	0.0333	1.0000	-0.5233	0.5900
1_Linear	1_Tree	0.0000	1.0000	-0.5248	0.5248
1_Linear	2_Linear	0.0667	1.0000	-0.4582	0.5915
1_Linear	2_Tail	-0.1333	0.9997	-0.6900	0.4233
1_Linear	2_Tree	-0.3333	0.6182	-0.8582	0.1915
1_Linear	3_Linear	-0.1333	0.9995	-0.6582	0.3915
1_Linear	3_Tail	0.0333	1.0000	-0.5233	0.5900
1_Linear	3_Tree	-0.6667	**0.0024**	-1.1915	-0.1418
1_Linear	4_Linear	-0.1333	0.9995	-0.6582	0.3915
1_Linear	4_Tail	0.0333	1.0000	-0.5233	0.5900
1_Linear	4_Tree	-0.5333	**0.0427**	-1.0582	-0.0085
1_Tail	1_Tree	-0.0333	1.0000	-0.5900	0.5233
1_Tail	2_Linear	0.0333	1.0000	-0.5233	0.5900
1_Tail	2_Tail	-0.1667	0.9985	-0.7534	0.4201
1_Tail	2_Tree	-0.3667	0.5624	-0.9233	0.1900
1_Tail	3_Linear	-0.1667	0.9977	-0.7233	0.3900
1_Tail	3_Tail	0.0000	1.0000	-0.5868	0.5868
1_Tail	3_Tree	-0.7000	**0.0028**	-1.2567	-0.1433
1_Tail	4_Linear	-0.1667	0.9977	-0.7233	0.3900
1_Tail	4_Tail	0.0000	1.0000	-0.5868	0.5868
1_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100

Continued on next page

Table A.3: (Continued) Tukey HSD Results for Correctness

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Tree	2_Linear	0.0667	1.0000	-0.4582	0.5915
1_Tree	2_Tail	-0.1333	0.9997	-0.6900	0.4233
1_Tree	2_Tree	-0.3333	0.6182	-0.8582	0.1915
1_Tree	3_Linear	-0.1333	0.9995	-0.6582	0.3915
1_Tree	3_Tail	0.0333	1.0000	-0.5233	0.5900
1_Tree	3_Tree	-0.6667	**0.0024**	-1.1915	-0.1418
1_Tree	4_Linear	-0.1333	0.9995	-0.6582	0.3915
1_Tree	4_Tail	0.0333	1.0000	-0.5233	0.5900
1_Tree	4_Tree	-0.5333	**0.0427**	-1.0582	-0.0085
2_Linear	2_Tail	-0.2000	0.9889	-0.7567	0.3567
2_Linear	2_Tree	-0.4000	0.3301	-0.9248	0.1248
2_Linear	3_Linear	-0.2000	0.9823	-0.7248	0.3248
2_Linear	3_Tail	-0.0333	1.0000	-0.5900	0.5233
2_Linear	3_Tree	-0.7333	**0.0005**	-1.2582	-0.2085
2_Linear	4_Linear	-0.2000	0.9823	-0.7248	0.3248
2_Linear	4_Tail	-0.0333	1.0000	-0.5900	0.5233
2_Linear	4_Tree	-0.6000	**0.0111**	-1.1248	-0.0752
2_Tail	2_Tree	-0.2000	0.9889	-0.7567	0.3567
2_Tail	3_Linear	0.0000	1.0000	-0.5567	0.5567
2_Tail	3_Tail	0.1667	0.9985	-0.4201	0.7534
2_Tail	3_Tree	-0.5333	0.0740	-1.0900	0.0233
2_Tail	4_Linear	0.0000	1.0000	-0.5567	0.5567
2_Tail	4_Tail	0.1667	0.9985	-0.4201	0.7534
2_Tail	4_Tree	-0.4000	0.4228	-0.9567	0.1567
2_Tree	3_Linear	0.2000	0.9823	-0.3248	0.7248
2_Tree	3_Tail	0.3667	0.5624	-0.1900	0.9233
2_Tree	3_Tree	-0.3333	0.6182	-0.8582	0.1915
2_Tree	4_Linear	0.2000	0.9823	-0.3248	0.7248
2_Tree	4_Tail	0.3667	0.5624	-0.1900	0.9233
2_Tree	4_Tree	-0.2000	0.9823	-0.7248	0.3248
3_Linear	3_Tail	0.1667	0.9977	-0.3900	0.7233
3_Linear	3_Tree	-0.5333	**0.0427**	-1.0582	-0.0085

Continued on next page

Table A.3: (Continued) Tukey HSD Results for Correctness

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
3_Linear	4_Linear	0.0000	1.0000	-0.5248	0.5248
3_Linear	4_Tail	0.1667	0.9977	-0.3900	0.7233
3_Linear	4_Tree	-0.4000	0.3301	-0.9248	0.1248
3_Tail	3_Tree	-0.7000	**0.0028**	-1.2567	-0.1433
3_Tail	4_Linear	-0.1667	0.9977	-0.7233	0.3900
3_Tail	4_Tail	0.0000	1.0000	-0.5868	0.5868
3_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100
3_Tree	4_Linear	0.5333	**0.0427**	0.0085	1.0582
3_Tree	4_Tail	0.7000	**0.0028**	0.1433	1.2567
3_Tree	4_Tree	0.1333	0.9995	-0.3915	0.6582
4_Linear	4_Tail	0.1667	0.9977	-0.3900	0.7233
4_Linear	4_Tree	-0.4000	0.3301	-0.9248	0.1248
4_Tail	4_Tree	-0.5667	**0.0420**	-1.1233	-0.0100

Table A.4: Tukey HSD Results for Response Time

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Linear	1_Tail	27.0681	0.9840	-44.9002	99.0363
1_Linear	1_Tree	-1.3691	1.0000	-69.2214	66.4832
1_Linear	2_Linear	7.6627	1.0000	-60.1896	75.5150
1_Linear	2_Tail	31.5161	0.9505	-40.4522	103.4843
1_Linear	2_Tree	25.1915	0.9855	-42.6608	93.0438
1_Linear	3_Linear	5.3551	1.0000	-62.4972	73.2074
1_Linear	3_Tail	19.1298	0.9992	-52.8384	91.0980
1_Linear	3_Tree	74.7007	**0.0177**	6.8484	142.5530
1_Linear	4_Linear	45.5182	0.5335	-22.3341	113.3705
1_Linear	4_Tail	62.0566	0.1656	-9.9116	134.0249
1_Linear	4_Tree	66.9403	0.0569	-0.9120	134.7926
1_Tail	1_Tree	-28.4371	0.9766	-100.4054	43.5311
1_Tail	2_Linear	-19.4053	0.9991	-91.3736	52.5629
1_Tail	2_Tail	4.4480	1.0000	-71.4132	80.3092

Continued on next page

Table A.4: (Continued) Tukey HSD Results for Response Time

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Tail	2_Tree	-1.8766	1.0000	-73.8448	70.0916
1_Tail	3_Linear	-21.7130	0.9975	-93.6812	50.2552
1_Tail	3_Tail	-7.9382	1.0000	-83.7994	67.9229
1_Tail	3_Tree	47.6326	0.5549	-24.3356	119.6008
1_Tail	4_Linear	18.4501	0.9994	-53.5181	90.4184
1_Tail	4_Tail	34.9886	0.9298	-40.8726	110.8497
1_Tail	4_Tree	39.8723	0.7942	-32.0960	111.8405
1_Tree	2_Linear	9.0318	1.0000	-58.8205	76.8841
1_Tree	2_Tail	32.8851	0.9340	-39.0831	104.8534
1_Tree	2_Tree	26.5605	0.9782	-41.2918	94.4128
1_Tree	3_Linear	6.7241	1.0000	-61.1282	74.5764
1_Tree	3_Tail	20.4989	0.9985	-51.4693	92.4671
1_Tree	3_Tree	76.0697	**0.0142**	8.2174	143.9220
1_Tree	4_Linear	46.8873	0.4860	-20.9650	114.7396
1_Tree	4_Tail	63.4257	0.1427	-8.5425	135.3939
1_Tree	4_Tree	68.3094	**0.0468**	0.4571	136.1617
2_Linear	2_Tail	23.8533	0.9943	-48.1149	95.8216
2_Linear	2_Tree	17.5287	0.9994	-50.3236	85.3810
2_Linear	3_Linear	-2.3077	1.0000	-70.1600	65.5446
2_Linear	3_Tail	11.4671	1.0000	-60.5011	83.4353
2_Linear	3_Tree	67.0379	0.0561	-0.8144	134.8902
2_Linear	4_Linear	37.8555	0.7867	-29.9968	105.7078
2_Linear	4_Tail	54.3939	0.3429	-17.5743	126.3621
2_Linear	4_Tree	59.2776	0.1516	-8.5747	127.1299
2_Tail	2_Tree	-6.3246	1.0000	-78.2928	65.6436
2_Tail	3_Linear	-26.1610	0.9878	-98.1292	45.8072
2_Tail	3_Tail	-12.3863	1.0000	-88.2474	63.4749
2_Tail	3_Tree	43.1846	0.6987	-28.7836	115.1528
2_Tail	4_Linear	14.0021	1.0000	-57.9661	85.9704
2_Tail	4_Tail	30.5406	0.9730	-45.3206	106.4017
2_Tail	4_Tree	35.4243	0.8940	-36.5440	107.3925
2_Tree	3_Linear	-19.8364	0.9981	-87.6887	48.0159

Continued on next page

Table A.4: (Continued) Tukey HSD Results for Response Time

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
2_Tree	3_Tail	-6.0617	1.0000	-78.0299	65.9066
2_Tree	3_Tree	49.5092	0.3983	-18.3431	117.3615
2_Tree	4_Linear	20.3267	0.9976	-47.5256	88.1790
2_Tree	4_Tail	36.8652	0.8658	-35.1030	108.8334
2_Tree	4_Tree	41.7489	0.6643	-26.1034	109.6012
3_Linear	3_Tail	13.7747	1.0000	-58.1935	85.7430
3_Linear	3_Tree	69.3456	**0.0403**	1.4933	137.1979
3_Linear	4_Linear	40.1631	0.7165	-27.6892	108.0154
3_Linear	4_Tail	56.7016	0.2809	-15.2666	128.6698
3_Linear	4_Tree	61.5853	0.1150	-6.2670	129.4376
3_Tail	3_Tree	55.5708	0.3104	-16.3974	127.5391
3_Tail	4_Linear	26.3884	0.9869	-45.5798	98.3566
3_Tail	4_Tail	42.9268	0.7711	-32.9343	118.7880
3_Tail	4_Tree	47.8105	0.5490	-24.1577	119.7787
3_Tree	4_Linear	-29.1825	0.9564	-97.0348	38.6698
3_Tree	4_Tail	-12.6440	1.0000	-84.6122	59.3242
3_Tree	4_Tree	-7.7603	1.0000	-75.6126	60.0920
4_Linear	4_Tail	16.5384	0.9998	-55.4298	88.5067
4_Linear	4_Tree	21.4221	0.9963	-46.4302	89.2744
4_Tail	4_Tree	4.8837	1.0000	-67.0845	76.8519

Table A.5: Tukey HSD Results for Fixation Count

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Linear	1_Tail	110.4333	0.9326	-130.5385	351.4052
1_Linear	1_Tree	-1.7333	1.0000	-228.9238	225.4571
1_Linear	2_Linear	6.4667	1.0000	-220.7238	233.6571
1_Linear	2_Tail	87.9333	0.9874	-153.0385	328.9052
1_Linear	2_Tree	112.4000	0.8905	-114.7905	339.5905
1_Linear	3_Linear	7.4667	1.0000	-219.7238	234.6571
1_Linear	3_Tail	76.9333	0.9959	-164.0385	317.9052

Continued on next page

Table A.5: (Continued) Tukey HSD Results for Fixation Count

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
1_Linear	3_Tree	252.0667	**0.0162**	24.8762	479.2571
1_Linear	4_Linear	1.9333	1.0000	-233.8333	237.7000
1_Linear	4_Tail	214.5167	0.1327	-26.4552	455.4885
1_Linear	4_Tree	216.0000	0.0791	-11.1905	443.1905
1_Tail	1_Tree	-112.1667	0.9253	-353.1385	128.8052
1_Tail	2_Linear	-103.9667	0.9553	-344.9385	137.0052
1_Tail	2_Tail	-22.5000	1.0000	-276.5067	231.5067
1_Tail	2_Tree	1.9667	1.0000	-239.0052	242.9385
1_Tail	3_Linear	-102.9667	0.9583	-343.9385	138.0052
1_Tail	3_Tail	-33.5000	1.0000	-287.5067	220.5067
1_Tail	3_Tree	141.6333	0.7253	-99.3385	382.6052
1_Tail	4_Linear	-108.5000	0.9522	-357.5740	140.5740
1_Tail	4_Tail	104.0833	0.9692	-149.9233	358.0900
1_Tail	4_Tree	105.5667	0.9503	-135.4052	346.5385
1_Tree	2_Linear	8.2000	1.0000	-218.9905	235.3905
1_Tree	2_Tail	89.6667	0.9852	-151.3052	330.6385
1_Tree	2_Tree	114.1333	0.8800	-113.0571	341.3238
1_Tree	3_Linear	9.2000	1.0000	-217.9905	236.3905
1_Tree	3_Tail	78.6667	0.9950	-162.3052	319.6385
1_Tree	3_Tree	253.8000	**0.0149**	26.6095	480.9905
1_Tree	4_Linear	3.6667	1.0000	-232.1000	239.4333
1_Tree	4_Tail	216.2500	0.1252	-24.7219	457.2219
1_Tree	4_Tree	217.7333	0.0738	-9.4571	444.9238
2_Linear	2_Tail	81.4667	0.9933	-159.5052	322.4385
2_Linear	2_Tree	105.9333	0.9245	-121.2571	333.1238
2_Linear	3_Linear	1.0000	1.0000	-226.1905	228.1905
2_Linear	3_Tail	70.4667	0.9981	-170.5052	311.4385
2_Linear	3_Tree	245.6000	**0.0220**	18.4095	472.7905
2_Linear	4_Linear	-4.5333	1.0000	-240.3000	231.2333
2_Linear	4_Tail	208.0500	0.1641	-32.9219	449.0219
2_Linear	4_Tree	209.5333	0.1016	-17.6571	436.7238

Continued on next page

Table A.5: (Continued) Tukey HSD Results for Fixation Count

Group 1	Group 2	Mean Difference	P-value	Lower	Upper
2_Tail	2_Tree	24.4667	1.0000	-216.5052	265.4385
2_Tail	3_Linear	-80.4667	0.9939	-321.4385	160.5052
2_Tail	3_Tail	-11.0000	1.0000	-265.0067	243.0067
2_Tail	3_Tree	164.1333	0.5089	-76.8385	405.1052
2_Tail	4_Linear	-86.0000	0.9920	-335.0740	163.0740
2_Tail	4_Tail	126.5833	0.8856	-127.4233	380.5900
2_Tail	4_Tree	128.0667	0.8346	-112.9052	369.0385
2_Tree	3_Linear	-104.9333	0.9290	-332.1238	122.2571
2_Tree	3_Tail	-35.4667	1.0000	-276.4385	205.5052
2_Tree	3_Tree	139.6667	0.6652	-87.5238	366.8571
2_Tree	4_Linear	-110.4667	0.9221	-346.2333	125.3000
2_Tree	4_Tail	102.1167	0.9607	-138.8552	343.0885
2_Tree	4_Tree	103.6000	0.9348	-123.5905	330.7905
3_Linear	3_Tail	69.4667	0.9983	-171.5052	310.4385
3_Linear	3_Tree	244.6000	**0.0230**	17.4095	471.7905
3_Linear	4_Linear	-5.5333	1.0000	-241.3000	230.2333
3_Linear	4_Tail	207.0500	0.1695	-33.9219	448.0219
3_Linear	4_Tree	208.5333	0.1055	-18.6571	435.7238
3_Tail	3_Tree	175.1333	0.4044	-65.8385	416.1052
3_Tail	4_Linear	-75.0000	0.9975	-324.0740	174.0740
3_Tail	4_Tail	137.5833	0.8168	-116.4233	391.5900
3_Tail	4_Tree	139.0667	0.7478	-101.9052	380.0385
3_Tree	4_Linear	-250.1333	**0.0271**	-485.9000	-14.3667
3_Tree	4_Tail	-37.5500	1.0000	-278.5219	203.4219
3_Tree	4_Tree	-36.0667	1.0000	-263.2571	191.1238
4_Linear	4_Tail	212.5833	0.1770	-36.4907	461.6573
4_Linear	4_Tree	214.0667	0.1147	-21.7000	449.8333
4_Tail	4_Tree	1.4833	1.0000	-239.4885	242.4552

B

Appendix: Code Snippets

The following sections list the code snippets we showed to participants with each possible input size marked as a comment next to the declaration of the input variable.

b.1 Linear Recursion

Listing B.1: Calculation of the Factorial, Source: https://www.tu-chemnitz.de/informatik/ST/publications/papers/PPIG_2021.pdf

```
public static void main(String[] args) {
    int n = 2;  // 3, 4, 5
    System.out.println(method(n));
}

static int method(int n) {
    if (n == 0)
        return 1;
    return n * method(n - 1);
}
```

Listing B.2: Conversion of a Decimal Number to Its Binary Representation, Source: https://www.tu-chemnitz.de/informatik/ST/publications/papers/PPIG_2021.pdf

```
public static void main(String[] args) {
    int n = 3;  // 5, 8, 17
    method(n);
}

static void method(int n){
    if (n == 0)
        return;
    method(n / 2);
    System.out.print(n % 2);
}
```

Listing B.3: Calculation of the Triangular Number, Source: Self-Written

```
public static void main(String[] args) {
    int n = 3;  // 4, 5, 6
    System.out.println(method(n));
}

static int method(int n) {
    if (n == 1)
        return 1;
    return n + method(n - 1);
}
```

Listing B.4: Summary of the Digits of a Number, Source: Self-Written

```
public static void main(String[] args) {
    int n = 523;  // 1523, 41523, 341523
    System.out.print(method(n));
}

static int method(int n) {
    if (n == 0)
        return 0;
    return n % 10 + method(n / 10);
}
```

Listing B.5: Calculation of the Logarithm Ignoring Float-Numbers, Source: Self-Written

```
public static void main(String[] args){
    int n = 5; // 9, 17, 33
    System.out.println(method(n));
}

static int method(int n){
    if (n == 1)
        return 0;
    else
        return 1 + method(n / 2);
}
```

b.2 Linear Recursion: Tail Recursion

Listing B.6: Calculation of the Prime Factors, Source: https://www.tu-chemnitz.de/informatik/ST/publications/papers/PPIG_2021.pdf

```
public static void main(String[] args) {
    int n = 4;  // 8, 16, 32
    method(n);
}

static void method(int n){
    int i = 2;
    if (n == 1)
        return;
    while (n % i != 0)
        i++;
    System.out.print(i + " ");
    method(n / i);
}
```

Listing B.7: Reversing a Number, Source: Self-Written

```
public static void main(String[] args) {
    int n = 125;  // 1253, 12538, 125386
    method(n);
}

static void method(int n) {
    if (n == 0)
        return;
    System.out.print(n % 10);
    method(n / 10);
}
```

Listing B.8: Test If a Number Is a Palindrome, Source: Self-Written

```
public static void main(String[] args) {
    String n = "765421245367";  // 765421243567, 765421234567, 765421324567
    System.out.println(method(n));
}

static boolean method(String n) {
    if (n.length() == 1) {
        return true;
    }

    if (n.charAt(0) == n.charAt(n.length() - 1)) {
        return method(n.substring(1, n.length() - 1));
    } else {
        return false;
    }
}
```

Listing B.9: Countdown, Source: Self-Written

```
public static void main(String[] args) {
    int n = 2;  // 3, 4, 5
    method(n);
}

static void method(int n) {
    System.out.println(n);
    if (n > 0)
        method(n - 1);
}
```

Listing B.10: Test If 2 Adjacent Numbers Sum Up to 5, Source: Self-Written

```
public static void main(String[] args){
    int n = 2344;  // 23442, 234425, 2344251
    method(n);
}

static void method(int n) {
    int a = n % 10;
    int b = (n % 100) / 10;

    if (b == 0)
        return;
    if (a + b == 5){
        System.out.print(b);
        System.out.print(a);
        return;
    }
    method(n / 10);
}
```

B.3 Tree Recursion

Listing B.11: Calculation of the Fibonacci Sequence, Source:
https://www.tu-chemnitz.de/informatik/ST/publications/papers/PPIG_2021.pdf

```
public static void main(String[] args) {
    int n = 2;  // 3, 4, 5
    System.out.println(method(n));
}

static int method(int n) {
    if (n <= 1)
        return n;
    return method(n - 1) + method(n - 2);
}
```

Listing B.12: Sequence Inspired by the Towers of Hanoi Problem, Source: Self-Written

```
public static void main(String[] args) {
    int n = 2;  // 3, 4, 5
    method(n, 'A', 'B', 'C');
}

static void method(int n, char a, char b, char c) {
    if (n > 0) {
        method(n - 1, a, c, b);
        System.out.print(n);
        method(n - 1, c, b, a);
    }
}
```

Listing B.13: Sequence, Inspired By: https://www.scholarhat.com/tutorial/c/recursion-in-c

```
public static void main(String[] args) {
    int n = 2;  // 3, 4, 5
    method(n);
}

static void method(int n) {
    if (n > 0) {
        System.out.println(n);
        method(n - 1);
        method(n - 1);
    }
}
```

Listing B.14: Calculation of the Largest Digit, Source: Self-Written

```
public static void main(String[] args){
    String n = "7984";  // 49874, 7498746, 37498746
    System.out.println(method(n));
}

static int method(String n){
    int a, b;
    switch (n.length()) {
        case 1:
            return Integer.parseInt(n);
        case 2:
            a = Character.getNumericValue(n.charAt(0));
            b = Character.getNumericValue(n.charAt(1));
            break;
        default:
            a = method(n.substring(0, n.length() / 2));
            b = method(n.substring(n.length() / 2, n.length()));
    }
    return a > b ? a : b;
}
```

Listing B.15: Sequence, Source:
https://www.geeksforgeeks.org/practice-questions-for-recursion-set-6/

```
public static void main(String[] args) {
    String n = "1";  // 12, 123, 1234
    method(n);
}

static void method(String n)
{
    if(n.length() == 0)
        return;

    method(n.substring(1));
    method(n.substring(1));
    System.out.print(n.charAt(0));
}
```

B.4 Nested Recursion

Listing B.16: Simple Calculation for the Participants to Warm-Up, Source: Self-Written

```
public static void main(String[] args) {
    int n = 16;
    System.out.print(fun(n));
}

static int fun(int n) {
    if (n > 20)
        return n - 3;
    return fun(fun(n + 5));
}
```

B.5 List of Presented Snippets

The table below illustrates the sequence in which each code snippet was presented, along with their respective input sizes. Each row corresponds to the order of snippets assigned to an individual participant.

Listing B.17: Code Sequence Assigned to Participants

```
D4,O1,N1,G3,C1,B4,P4,I2,Q1,A4,S2,R3,H3,L3,J3,
L4,R4,J4,I3,N2,B1,G4,O2,Q2,P1,D1,S3,A1,C2,H4,
D2,A2,O3,B2,G1,L1,H1,S4,I4,P2,C3,J1,R1,N3,Q3,
I1,G2,H2,J2,C4,B3,D3,L2,Q4,S1,P3,O4,N4,R2,A3,
C1,A4,J3,R3,S2,H3,D4,Q1,G3,I2,N1,O1,L3,P4,B4,
I3,H4,O2,D1,Q2,G4,A1,P1,N2,J4,C2,S3,B1,R4,L4,
P2,O3,R1,C3,J1,A2,L1,S4,Q3,N3,H1,I4,G1,B2,D2,
C4,B3,A3,L2,N4,I1,P3,Q4,G2,J2,D3,H2,O4,R2,S1,
D4,A4,G3,B4,J3,R3,P4,L3,H3,S2,N1,Q1,C1,I2,O1,
H4,P1,N2,S3,A1,L4,G4,B1,Q2,D1,J4,O2,R4,I3,C2,
O3,I4,R1,N3,L1,A2,H1,Q3,P2,S4,C3,D2,J1,B2,G1,
A3,P3,N4,D3,R2,H2,J2,O4,C4,G2,L2,B3,I1,Q4,S1,
```

Statement on the Usage of Generative Digital Assistants

For this thesis, the following generative digital assistants have been used: We have used OpenAI's *ChatGPT*[1] for ideation, rephrasing, and code completion and refactoring in our evaluation scripts. *Grammarly*[2] was employed to ensure grammatical accuracy and improve overall readability. Additionally, *Quillbot*[3] was used for text rephrasing and paraphrasing specific sections to enhance linguistic variety.

We are fully aware of the potential risks associated with relying on such tools. Therefore, these tools were applied thoughtfully, with critical evaluation of their outputs to maintain the academic integrity and originality of the work. We are aware of the potential dangers of using these tools and have used them sensibly with caution and with critical thinking.

1 OpenAI, *ChatGPT (2024 version)*, accessed November 28, 2024, https://chat.openai.com

2 Grammarly Inc., *Grammarly Writing Assistant (2024 version)*, accessed November 28, 2024, https://www.grammarly.com.

3 Quillbot Inc., *Quillbot AI Paraphrasing Tool (2024 version)*, accessed November 28, 2024, https://www.quillbot.com.

Bibliography

[1] Arooba Aqeel, Norman Peitek, Sven Apel, Jonas Mucke, and Janet Siegmund. "Understanding Comprehension of Iterative and Recursive Programs with Remote Eye Tracking." In: *PPIG*. Psychology of Programming Interest Group, 2021.

[2] Jennifer Bauer, Janet Siegmund, Norman Peitek, Johannes C. Hofmeister, and Sven Apel. "Indentation: simply a matter of style or support for program comprehension?" In: *ICPC*. IEEE / ACM, 2019, pp. 154–164.

[3] Roman Bednarik and Markku Tukiainen. "An eye-tracking methodology for characterizing program comprehension processes." In: *ETRA*. ACM, 2006, pp. 125–132.

[4] Yoav Benjamini and Yosef Hochberg. "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing." In: *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (1995), pp. 289–300.

[5] Haiqin Chen, Paul De Boeck, Matthew Grady, Chien-Lin Yang, and David Waldschmidt. "Curvilinear dependency of response accuracy on response time in cognitive tests." In: *Intelligence* 69 (2018), pp. 16–23.

[6] Norman Cliff. "Dominance statistics: Ordinal analyses to answer ordinal questions." In: *Psychological Bulletin* 114 (1993), 494–509.

[7] Graham Cooper. "Cognitive load theory as an aid for instructional design." In: *Australasian Journal of Educational Technology* 6.2 (1990).

[8] Chris Donkin, Daniel R. Little, and Joseph W. Houpt. "Assessing the speed-accuracy trade-off effect on the capacity of information processing." In: *Journal of Experimental Psychology: Human Perception and Performance* 40.3 (2014), 1183–1202.

[9] Timon Dörzapf, Norman Peitek, Marvin Wyrich, and Sven Apel. "Data Analysis Tools Affect Outcomes of Eye-Tracking Studies." In: *ESEM*. ACM, 2024, pp. 96–106.

[10] Andrew T Duchowski. *Eye tracking methodology*. Springer, 2007.

[11] Olive Jean Dunn. "Multiple Comparisons Using Rank Sums." In: *Technometrics* 6.3 (1964), pp. 241–252.

[12] N. Ernst. *cliffsDelta*. 2024. URL: https://github.com/neilernst/cliffsDelta.

[13] Lucian José Gonçales, Kleinner Farias, and Bruno C. da Silva. "Measuring the cognitive load of software developers: An extended Systematic Mapping Study." In: *Information and Software Technology* 136 (2021).

[14] Michael T. Goodrich, Roberto Tamassia, and David M. Mount. *Data structures and algorithms in C++*. Wiley, 2004.

[15] Yuxuan Guo, Sebastian Pannasch, Jens R Helmert, and Aleksandra Kaszowska. "Ambient and focal attention during complex problem-solving: preliminary evidence from real-world eye movement data." In: *Front. Psychol.* 15 (2024).

[16] Bruria Haberman and Haim Averbuch. "The case of base cases: why are they so difficult to recognize? student difficulties with recursion." In: *ITiCSE*. ACM, 2002, pp. 84–88.

[17] Marcel A Just and Patricia A Carpenter. "A theory of reading: From eye fixations to comprehension." In: *Psychol. Rev.* 87.4 (1980), pp. 329–354.

[18] Owen Kaser, C. R. Ramakrishnan, and Shaunak Pawagi. "On the Conversion of Indirect to Direct Recursion." In: *LOPLAS* 2.1-4 (1993), pp. 151–164.

[19] William H. Kruskal and W. Allen Wallis. "Use of Ranks in One-Criterion Variance Analysis." In: *Journal of the American Statistical Association* 47.260 (1952), 583–621.

[20] Elynn Lee, Victoria Shan, Bradley Beth, and Calvin Lin. "A structured approach to teaching recursion using cargo-bot." In: *ICER*. ACM, 2014, pp. 59–66.

[21] Colleen M. Lewis. "Exploring variation in students' correct traces of linear recursion." In: *ICER*. ACM, 2014, pp. 67–74.

[22] Anneliese von Mayrhauser and A. Marie Vans. "From program comprehension to tool requirements for an industrial environment." In: *WPC*. IEEE, 1993, pp. 78–86.

[23] Renée A. McCauley, Brian Hanks, Sue Fitzgerald, and Laurie Murphy. "Recursion vs. Iteration: An Empirical Study of Comprehension Revisited." In: *SIGCSE*. ACM, 2015, pp. 350–355.

[24] Renée McCauley, Scott Grissom, Sue Fitzgerald, and Laurie Murphy. "Teaching and learning recursive programming: a review of the research literature." In: *Comput. Sci. Educ.* 25.1 (2015), pp. 37–66.

[25] Dauglas Montgomery and Cahyono St. *Design and Analysis of Experiments, 9th Edition*. John Wiley & Sons, Inc., 2022.

[26] Alberto S. Núñez-Varela, Héctor G. Pérez-González, Francisco E. Martínez-Pérez, and Daniela Esqueda-Contreras. "A Study And Experimental Assessment Of The Cognitive Weight, Base Of The Cognitive Metrics." In: *2019 7th International Conference in Software Engineering Research and Innovation (CONISOFT)*. 2019, pp. 173–180.

[27] Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. "A Survey on the Usage of Eye-Tracking in Computer Programming." In: *ACM Comput. Surv.* 51.1 (2018), 5:1–5:58.

[28] Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, and Sven Apel. "Correlates of Programmer Efficacy and Their Link to Experience: A Combined EEG and Eye-Tracking Study." In: *CoRR* abs/2303.07071 (2023).

[29] Norman Peitek, Janet Siegmund, and Sven Apel. "What Drives the Reading Order of Programmers?: An Eye Tracking Study." In: *ICPC*. ACM, 2020, pp. 342–353.

[30] Keith Rayner. "Eye movements in reading and information processing: 20 years of research." In: *Psychol. Bull.* 124.3 (1998), pp. 372–422.

[31] Christian Rinderknecht. "A Survey on Teaching and Learning Recursive Programming." In: *Informatics Educ.* 13.1 (2014), pp. 87–119.

[32] Eric S. Roberts. *Thinking recursively*. Wiley, 1986.

[33] S. S. SHAPIRO and M. B. WILK. "An analysis of variance test for normality (complete samples)†." In: *Biometrika* 52.3-4 (1965), pp. 591–611.

[34] Ivonne Schröter, Jacob Krüger, Janet Siegmund, and Thomas Leich. "Comprehending studies on program comprehension." In: *ICPC*. IEEE Computer Society, 2017, pp. 308–311.

[35] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha E. Crosby. "A practical guide on conducting eye tracking studies in software engineering." In: *Empir. Softw. Eng.* 25 (2020), pp. 3128–3174.

[36] Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. "A systematic literature review on the usage of eye-tracking in software engineering." In: *Inf. Softw. Technol.* 67 (2015), pp. 79–107.

[37] Janet Siegmund. "Program Comprehension: Past, Present, and Future." In: *FOSE@SANER*. IEEE Computer Society, 2016, pp. 13–20.

[38] Seabold Skipper and Perktold Josef. "statsmodels: Econometric and statistical modeling with python." In: *9th Python in Science Conference* (2010).

[39] Gert Smolka. *Programmierung - eine Einführung in die Informatik mit Standard ML*. Oldenbourg, 2008.

[40] Zéphyrin Soh, Foutse Khomh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. "Noise in Mylyn interaction traces and its impact on developers and recommendation systems." In: *Empir. Softw. Eng.* 23.2 (2018), pp. 645–692.

[41] John Sweller. "Cognitive Load During Problem Solving: Effects on Learning." In: *Cognitive Science* 12.2 (1988), pp. 257–285.

[42] Maksim Terpilowski. "scikit-posthocs: Pairwise multiple comparison tests in Python." In: *The Journal of Open Source Software* 4.36 (2019), p. 1169.

[43] Eva Thilderkvist and Felix Dobslaw. "On current limitations of online eye-tracking to study the visual processing of source code." In: *Inf. Softw. Technol.* 174 (2024), p. 107502.

[44] John W. Tukey. "Comparing Individual Means in the Analysis of Variance." In: *Biometrics* 5.2 (1949), pp. 99–114.

[45] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272.

[46] Susan Wiedenbeck. "Learning Iteration and Recursion from Examples." In: *Int. J. Man Mach. Stud.* 30.1 (1989), pp. 1–22.

[47] Marvin Wyrich. "Source Code Comprehension: A Contemporary Definition and Conceptual Model for Empirical Investigation." In: *CoRR* abs/2310.11301 (2023).

[48] Marvin Wyrich, Justus Bogner, and Stefan Wagner. "40 Years of Designing Code Comprehension Experiments: A Systematic Mapping Study." In: *ACM Comput. Surv.* 56.4 (2024), 106:1–106:42.

[49] Johannes Zagermann, Ulrike Pfeil, and Harald Reiterer. "Studying Eye Movements as a Basis for Measuring Cognitive Load." In: *CHI Extended Abstracts*. ACM, 2018.