Bachelor's Thesis

Which Atoms Confuse Whom: A Comparative Eye Tracking Study In Program Comprehension

Andi Keraj

December 22, 2025

Advisors:

Annabelle Bergum	Chair of Software Engineering
Norman Peitek	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

Erklärung Statement

Saarbrücken, ______________ ____________________________
(Datum Date) (Unterschrift Signature)

Einverständniserklärung (optional) Declaration of Consent (optional)

Saarbrücken, ______________ ____________________________
(Datum Date) (Unterschrift Signature)

Abstract

Writing and understanding code are two fundamental skills every developer must master. Although both are important, understanding and comprehending written code is a cognitively demanding process. This process becomes more difficult when the code is misinterpretable due to confusing code patterns known as "atoms of confusion". Research shows that these patterns have a negative effect on developers' program comprehension, at times being the cause of security risks and serious system failures.

In our study, we compare the effect of different categories of atoms of confusion by analyzing eye-tracking data gathered from participants. We found that atoms of the categories conditional operators and pre/post in/decrement cause a significant increase in Area of Interest (AOI) fixation count, regressions, and AOI fixation duration when compared to other categories. This is a result of an increased visual and cognitive effort when presented with such atoms.

We investigate whether experience in programming plays a role in the confusion caused by the atoms. We discover that atoms of categories, conditional operators, scope indentation, and variable reassignment have the same effect on participants, independent of the experience. We found no statistically significant difference in the observed visual effort from the participants.

On the other hand, novice participants fixated much more on the code and on the AOI, focusing longer and regressing back to the AOI when encountering atoms in the pre/post in/decrement category.

Furthermore, we found a statistically significant difference between expert and intermediate participants in the observed number of AOI fixations and regression. Intermediate participants resulted in significantly lower AOI fixation counts and regressions compared to experts.

We explore the possibility of predicting the presence of an atom in code snippets by making use of machine learning. We train three machine learning classification models [1], and we evaluate their performance on test data. We found that, the SVM performed the best out of the three models, achieving a higher accuracy of predictions.

1 Soft margin Support Vector Machine (SVM), Extreme Gradient Boosting (XGBoost), Random Forest

Contents

List of Figures

List of Tables

Listings

Acronyms

AOC	Atoms of Confusion
IOCCC	International Obfuscated C Code Contest
GCC	GNU Compiler Collection
AOI	Area of Interest
SVM	Support Vector Machine
XGBoost	Extreme Gradient Boosting

1

Introduction

Program comprehension is a crucial aspect in the field of software development and maintenance. Understanding and extending upon code written by other developers is a very common activity in the professional world of code development. It is frequently seen that code in software projects contains code patterns that cause confusion [13]. These confusing code patterns are known as AOC [12].

AOC are small patterns that have been empirically validated to be difficult to hand-evaluate by developers [11]. They are very prevalent in software programs and projects, where they occur on average once every 23 lines [13]. AOC are usually a result of bad programming practice, but sometimes are also encouraged by coding standards [12]. The presence of AOC in code causes obstruction in the ability of developers to understand and comprehend the code [9–11, 13, 25]. It has been shown that such patterns hinder program comprehension, and when the AOC are removed from code, there is an increase in hand evaluation accuracy by 50% [12].

AOC not only hinder the developer's understanding of the program but also introduce bugs and faulty software implementations. These bugs can result in millions of dollars worth of damage. Examples like Apple's 'goto fail' SSL bug [4] allowed for an SSL man-in-the-middle vulnerability for all OSX and iOS devices. Furthermore, Ariane 5's crash and explosion [17] was caused by a floating point overflow bug that resulted in a $500 million crash. Such failures are not caused by faulty algorithms or malfunctioning systems, but rather very isolated, small errors at a semantic level that come from confusing and unintuitive constructs [12]. Avoiding these confusing lines of code allows for better program comprehension, fewer bugs, lower maintenance costs, and potentially preventing disasters like Ariane 5.

When it comes to program comprehension, eye-tracking methods are prevalent in many studies and research regarding the topic [1, 29, 31]. Eye-tracking metrics such as fixation count, fixation duration, and regression have been widely used in program comprehension research. These metrics are used to extract information regarding reading patterns [30], cognitive load [23], and task complexity [7]. When AOC are present in code, there is an increase in gaze transitions by 36.8% [10], suggesting that the atom is often the centre of attention as the subjects try to comprehend what the atom does or calculates. We also leverage these metrics to explore and investigate the research questions of this study.

1.1 Goal of this Thesis

The goal of this thesis is to investigate the effects of AOC on programmers' comprehension, focus, and understanding of the program by analyzing eye movement patterns and fixations on the code. While other studies conlcude that AOC hinder program comprehension and cause confusion to participants [9–12], there is a limited understanding of how different atoms cause different levels of confusion. In this thesis, we investigate which atoms inflict more fixations, longer gaze durations, and more regressions. Additionally, we determine whether previously identified atoms are inherently confusing due to their constructs or if they cause confusion due to the lack of knowledge or experience with the programming language. Comparing the eye movement behavior between novices and experts, we make the distinction between the confusion that originates from unfamiliarity and that caused by a complex construct.

By combining eye-tracking data with participants' performance metrics and demographic data, this thesis contributes by providing a deeper understanding of which atoms are problematic and more severe, for whom, and why. The benefit of these insights will allow for better programming practices, identification of new AOC, and code writing tools to minimize and avoid the occurrences of such confusing patterns.

2 Background

In this chapter, we provide information on important definitions and relevant content to this thesis. In section Section 2.1, we explain what program comprehension is, how it is defined, and how it was used in previous research. In section Section 2.2, we provide information on what AOC are, and we provide examples of such atoms. Section Section 2.3 of this chapter explains key eye-tracking terminologies.

2.1 Program Comprehension

Program comprehension is a key activity in understanding how a software system or a part of it works. Comprehension is a very time-consuming task where it is reported that developers spend about half their time comprehending programs during software maintenance [26].

Many studies have investigated the factors that affect developers' program comprehension, taking into account various measures that reflect the level of comprehension [1, 3, 22, 29, 37]. However, it is found that different measures that are used to analyze program comprehension do not correlate with each other.

In a study regarding the definition of program comprehension, Wyrich [43] states that since the dawn of research in program comprehension, there is no universally agreed-upon definition of program comprehension. The lack of a definition for such an important and researched term has led researchers to come up with different definitions. Shneiderman [38] for example, defines program comprehension as "the recognition of the overall function of the program, an understanding of intermediate level processes including program organization and comprehension of the function of each statement in a program". Pennington [32] states that "comprehension involves the assignment of meaning to a particular program, an accomplishment that requires the extensive application of specialized knowledge". Many other researchers have made their own definitions of program comprehension. This leads to the usage of different measures for analyzing program comprehension, and hence, it is often the case that they are not comparable.

In the paper, Wyrich [43] formulates a definition for source code comprehension instead of program comprehension to distinguish code comprehension from other objects in the diverse field of program comprehension. The definition is the following:

Definition 1 *Source code comprehension describes a person's intentional act and degree of accomplishment in inferring the meaning of code.*

Experience in Code Reading

Experience is a critical factor in a developer's ability to understand and write code efficiently. Many studies use years of experience [2, 19], education level [16, 27], or employment status [21] as an indicator for a developer's level of expertise. While these measures are logical to use, another measure of evaluating the knowledge and programming proficiency is efficacy.

Peitek et al. [29] introduced efficacy as an alternative measure to years of experience or education level. The research concluded that traditional experience measures do not accurately reflect program comprehension and writing skills, but instead, efficacy is a more viable and accurate option. Efficacy is evaluated based on the fraction of the number of correct tasks participants complete and the time it takes for completion [29].

2.2 Atoms of Confusion

AOC are small patterns empirically validated to be difficult to understand and hand-evaluate by developers [11]. AOC are vastly present in big projects [13]. In many studies regarding AOC, the initial set of atoms have been picked from the International Obfuscated C Code Contest (IOCCC). The code snippets are designed in such a way that they intentionally cause confusion to the developers. While some of the atoms obtained from IOCCC may not represent real-life programs, they demonstrate patterns that do cause confusion to developers in real life.

2.2.1 Categories of Atoms of Confusion

In this section, we will describe and show some of the AOC categories considered in our study. The categories of atoms that we consider are the ones that are found and discussed by Gopstein et al. [12].

Post-increment Listing 2.5 shows a small code snippet containing a variable assignment of $V1$ and R with an arithmetic statement. The confusing element in this snippet is that it is easy to overlook the fact that, first, the value of $V1$ and 1 will be added together, and only then will $V1$ be incremented. The value of R will be equal to 8.

Mis-indentation Indentation is a very important component in the semantics of the program. Listing 2.3 shows a code snippet which contains multiple `if` and `else` statements. In the snippet, we notice that the `else` statement belongs to the first `if` clause. This is not the case because the curly braces indicate that the `else` statement belongs to the nested `if` case, which can lead to incorrect evaluation of this program if not carefully inspected. This discussion assumes a brace-based language (e.g., `C`, `Java`), where indentation is not part of the formal syntax. In contrast, in languages such as `Python`, mis-indentation directly changes the program's control flow rather than only reducing developers' program comprehension.

Conditional operators Conditional operators are very important in control flow variables and constructs. Using multiple conditional operators in a statement while omitting parentheses can be confusing when evaluating a code snippet. Listing 2.4 shows the initializations of three boolean variables. In the next statement, we have the assignment of a variable to a Boolean statement. Here, it is unclear in which order the &&, !, and the || operators bind, which affects the evaluation of the statement.

Omitted curly braces Curly braces are important indicators of identifying statements that are part of for-loops,while-loops, and if clauses. They allow us to distinguish between statements that are part of these constructs and statements that are not. Listing 2.2 shows a `for` loop iterating over a variable $v1$. It is unclear if both of the $R++$ statements belong to the `for` loop or only one of them. This affects the end value of R.

Arithmetic operators binding Operator binding can sometimes be unclear when it comes to mathematical statements in programming languages. Listing 2.1 shows the binding of the variable R to the statement $3+9/3$. The lack of parentheses makes this snippet confusing, as the developers cannot be sure which operand is evaluated first. This yields two different results, which impact the understanding of the program by the developer.

Type conversion It is very common to convert an integer type to a float type (e.g., convert 3 to 3.0). What is not very common is converting an integer type to a byte. Listing 2.6 shows a variable assignment $V1$ to 288, and in the second line, we notice that this variable is cast to a byte. This line might cause confusion, as the developer viewing this code snippet might not know exactly what this line does. In `Java`, a byte is an 8-bit signed integer with a range of -128 to 12. Since the value of $V1$ is out of range for a byte, when it is cast to a byte, it will overflow using modulo 256 arithmetic. This will result in the value of $R = 32$.

Listing 2.1: Arithmetic operators binding

```
int R;
R = 3 + 9 / 3;
```

Listing 2.2: Omitted curly braces

```
int R = 4;
for (int V1 = 0; V1 < 3; V1++) R++; R++;
```

Listing 2.3: Mis-indentation

```
int V1 = 2;
int V2 = 8;
int R = 5;
if (V1 > 0) {
   if (V2 > 0) {
      R = R + 2;
}  else {
      R = R + 4;
}
}
```

Listing 2.4: Conditional Operators

```
boolean V1 = false;
boolean V2 = true;
boolean V3 = true;
boolean R = (V1 && !V2 || V3);
```

Listing 2.5: Post-increment

```
int V1 = 7;
int R = 1 + V1++;
```

Listing 2.6: Type conversion

```
int V1 = 288;
int R = (byte) V1;
```

2.3 Eye Tracking

Eye tracking is a powerful method that is used to study how programmers comprehend code [6, 40]. It provides crucial information in measuring visual attention and cognitive processing of programmers when observing a piece of code [18]. By analyzing where and how long and often a subject looks at a specific part of the code, it is possible to infer what parts of the code are mentally demanding [39]. To make this inference, the following eye-tracking metrics are used:

- **Fixations**: In eye tracking, a fixation is a period of time during which the gaze of an individual is relatively still, allowing them to process information at a specific point in space. The average fixation duration during code reading varies depending on the task assigned to the subjects, but typically a duration threshold of 100-200 ms is used to identify potential fixations [35].

- **Saccades**: Saccades are fast eye movements that shift the center of gaze from one part of the visual field to another. Saccades are very fast, and they can reach speeds up to 700 degrees/s [36].

- **Regressions**: Regressions in code reading refer to backwards eye movements, where the participants' eyes move back to previously fixated areas of interest in code. A high number of regressions during a code reading task indicates confusion and difficult code comprehension [9].

3

Related Work

In this chapter, we discuss and present previous work on atoms of confusion. We present the results and findings of the studies relevant to our work. We provide a timeline in Figure 3.1 of the key research on atoms of confusion throughout the years.

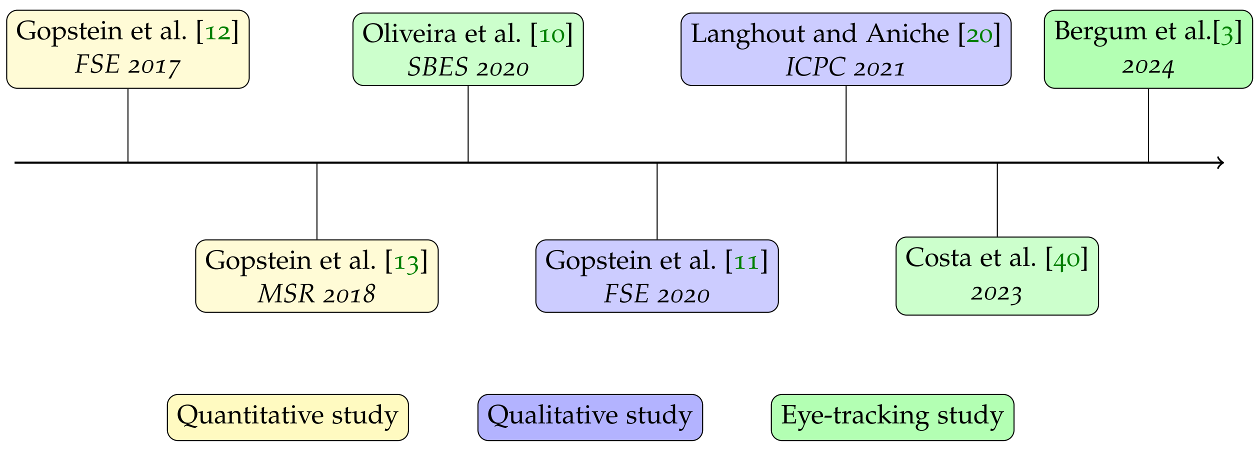

Figure 3.1: Timeline of related work on AOC and code comprehension.

3.1 Quantitative Studies

In one of the first studies on AOC, Gopstein et al. [12] conducted a controlled experiment, where participants were asked to hand-evaluate a set of code snippets with atoms and the clarified counterparts of each code snippet. They found that the error rate in code snippets with atoms was higher than in the clarified counterparts. They concluded that the mean correctness rate increased by more than 50% between code snippets with atoms and clarified code snippets [12].

In a different study, Gopstein et al. [13] found evidence that AOC are vastly prevalent in big software projects like the Linux kernel or GNU Compiler Collection (GCC). In these programs, AOC occur on average once every 23 lines. Additionally, in lines where an AOC is present, it is 1.13 times more likely to include comments than lines of code with no AOC. This evidence clearly tells us that AOC hinder code and their presence causes a significant decrease in program comprehension.

Although it is made clear that AOC negatively affect program understanding and comprehension, many previous studies were performed on subjects who are undergraduate students with limited experience in programming [12, 40]. This is an important fact to consider as the participants of these studies might not have sufficient knowledge regarding the programming language to understand complex constructs in detail.

Medeiros et al. [25] conducted a study investigating the presence of "misunderstanding code patterns" in 50 C open-source projects such as Apache, OpenSSL, and Python.

They found more than 109 thousand occurrences of 12 distinct atoms. The AOC that they included in the study involve *Multiple Initializations, Conditional Operator, Initialization in Conditions, Assignment as Value, Pointer Arithmetic, Post/Pre Increment, Operator Precedence, Dangling Else, Logic as Control Flow, Comma Operator, Reversed Subscript*.

They conducted a survey that included questions regarding the experience of developers with the atoms considered in the study. The survey was sent to 701 developers, and a total of 97 developers completed it. Their findings show that among the 12 atoms proposed in the survey, most of the developers agree that 6 out of 12 atoms might cause misunderstandings.

3.2 Qualitative studies

Gopstein et al. [11] performed a qualitative study on the effect of AOC on participants' program comprehension. They discovered that evaluating the correct output of an obfuscated program does not necessarily mean that the participant understood the program. They grouped the subjects' errors into four categories:

- **Unfamiliarity**: the subject had never seen the code construct before.
- **Misunderstanding**: the subject is aware of the construct, but has an incorrect mental model of the construct.
- **Language transfer**: the subjects make incorrect assumptions based on knowledge from other programming languages.
- **Attention**: the subject applies correct semantics, but forgets or ignores an important piece of the computation.

While previous studies conducted research on AOC in C programming snippets, Langhout and Aniche [20] used Java code snippets to perform their study.

Their experiment involved 132 Computer Science students and 14 AOC, which were derived from Gopstein et al. [12] work. The code snippets were translated to Java and made to be as similar to the version in C as possible, while also having clarified counterparts for each atom.

They found that participants are 4.6 up to 56 times more likely to make mistakes in code snippets affected by 7 out of 14 atoms. Furthermore, when faced with both versions of the code snippets, the participants stated that the code snippets affected by the atom of confusion were more confusing and/or less readable in 10 out of 14 studied atoms.

3.3 Eye-tracking Studies

Eye-tracking methods and metrics are used to analyze the effects of confusing code on participants' viewing of the code snippets. Oliveira et al. [10] conducted a study asking participants to evaluate the output of three programs with AOC and three programs without AOC. By using an eye tracker to capture eye movement data, they observed an increase of 36.8% in gaze transitions in programs with atoms. They found that, in the presence of an atom, the time required to understand code correctly increased by 43.02%.

Silva da Costa et al. [40] performed a study using eye-tracking on 128 novices where the participants were tasked with evaluating the output of code snippets in Java, C, and Python. They concluded that on clarified code snippets, the completion time and number of attempts was reduced by 38.6% and 28% respectively. This result provides further evidence that AOC negatively impact programmers understanding and comprehension.

These findings add another dimension to the research regarding AOC. While it is certain that atoms cause confusion and negatively affect comprehension among programmers, it should be considered that this confusion might arise from insufficient knowledge or experience with the programming language in the case of novices. We claim that, while some atoms may not be confusing to programming experts and developers, they cause confusion to novices (due to unfamiliar syntax or constructs).

4 Methodology

In this chapter, we introduce and discuss the main research questions of this thesis, along with the motivation behind them. Further in the following sections, we introduce the data, what it comprises, and how we utilize it in our investigation to answer the research questions.

4.1 Research Questions

This thesis answers the following research questions:

> **RQ1:** Do certain atoms cause more confusion than others, and if so, which ones?

Prior research has revealed that atoms hinder comprehension of participants looking at the programs [3, 9–12]. In these studies, atoms are often treated uniformly; however, some atoms can take more effort to comprehend or cause more confusion than other atoms. This extra effort causes participants to revisit the atoms more often and fixate for longer. By investigating which atoms require more visual attention and cognitive effort, while also taking into account the time and correctness of the code snippets' evaluation, we are able to distinguish which atoms cause more confusion to participants. This allows us to prioritise avoiding certain patterns that cause a higher level of confusion, to enhance readability and maintainability.

> **RQ2:** Do novices and experts experience different levels of confusion when encountering the same atom?

Previous eye-tracking studies have concluded that experts are naturally able to comprehend code at a much faster rate when compared to novices [29, 30]. Experts tend to read code less linearly and, by utilising prior knowledge and experience, they can scan code more selectively and efficiently. Novices, on the other hand, often read code more linearly, focusing on each line of code and its surface-level syntax. In some cases, atoms can appear confusing to novices, not because of a misleading construct, but because they are unfamiliar or not experienced enough with the programming language [11]. Analysing and finding a difference in eye movement behaviour between experts and novices can reveal whether certain atoms are confusing only to groups of people, when accounting for programming experience.

> **RQ3:** How accurately do confusion measurement metrics predict the likelihood of an atom being present in a code snippet?

In eye-tracking, metrics such as fixation duration are widely used in research as a measure of cognitive load [39]. Typically, when participants exhibit a higher fixation duration, fixation count on a particular element of the code shows that participants experience a higher cognitive load to understand and process the confusing atom. As demonstrated in previous studies [10, 40], atoms of confusion cause a significantly higher error rate in code evaluation among subjects, and in some cases, participants choose to give up on the evaluation task. We investigate the relationship between visual attention, error making, completion time, and correctness. Evaluating whether there is a correlation between these metrics and the participants' errors helps us identify new atoms and their level of impact.

4.2 Data Preparation and Manipulation

4.2.1 Description of Existing Dataset

This thesis builds upon the dataset introduced by Bergum et al. [3]. The eye-tracking data was recorded using a Tobii Pro Spectrum with a sampling rate of 1200 HZ, and calibration was done for each participant to ensure high accuracy. The data is already preprocessed and discussed in detail in the paper by Bergum et al. [3]. Additionally, we explain important procedures we follow to analyse the data.

The eye-tracking data is structured as follows: For each participant, the data is separated into different files based on the name of the snippet from which the data was gathered. An example of such data can be seen in Table 4.1. The file contains a timestamp of the fixation start and the fixation end. It contains the location of the fixation in the x and y axes as well as the radius of the fixation. In addition to these metrics, we have data regarding the duration of each fixation of the participant, the snippet name corresponding to these fixations, and the condition of the snippet. This condition indicates whether the snippet contains an AOC or not.

In addition to the raw eye-tracking data, there are two other data files. One contains demographic data of the participants, such as age group, gender, profession, years of programming, contact with different programming languages, and recent programming activity.

The other data file contains behavioural data of all participants, with information such as the start and end of the fixation, the total fixation duration for the snippet, the snippet name, and the answer that the participant gave for the snippet.

To work with the data, we made use of the `Pandas` $v2.3.3$ package [44] implemented in `Python`. We state the steps taken for each research question.

Table 4.1: Example of fixation eye-tracking data for participant 7 on snippet "4-obf-vo".

start	end	x	y	radius x	radius y	duration	Snippet	Participant	Condition
738.91	739.55	619.01	290	0.71	1.58	0.64	4-obf-vo	7	confusing
739.57	739.88	704.20	343	0.35	0.30	0.31	4-obf-vo	7	confusing
740.06	740.80	594.93	290	1.14	1.44	0.74	4-obf-vo	7	confusing
741.15	741.65	756.40	343	1.59	0.47	0.50	4-obf-vo	7	confusing
741.66	742.21	705.35	343	1.55	0.39	0.55	4-obf-vo	7	confusing
742.22	743.27	768.37	343	1.11	0.70	1.05	4-obf-vo	7	confusing
743.45	743.68	672.21	343	0.45	0.81	0.23	4-obf-vo	7	confusing
743.69	744.16	677.86	343	1.60	2.27	0.47	4-obf-vo	7	confusing
744.17	744.62	766.32	343	0.52	0.83	0.45	4-obf-vo	7	confusing

4.2.2 Analysis Procedure for RQ1: Categories of Atom of Confusion

RQ1: Do certain atoms cause more confusion than others, and if so, which ones?

To answer this research question, we statistically compare eye-tracking metrics like fixation count, AOI fixation count, AOI fixation duration, and regressions between each AOC category. To make this comparison, we derive these metrics from the existing eye-tracking data for each AOC category.

First, we iterate through the data of each participant and each snippet, calculating the metrics respectively. We create a dataframe for each metric where we list for each participant the corresponding value for that metric for each snippet. If a participant does not have any data for a specific snippet, we assign "NaN" to that entry.

The metrics are calculated in the following way:

Fixation count - the total fixation count on the whole snippet, calculated by taking the length of the data for the respective participant and snippet.

AOI fixation count - the total count of fixations that fall within the area of intereste (AOI), i.e the atom of confusion. Knowing the borders of the AOIs in pixels, we filter the fixation data to include only those entries whose fixations fall within this AOI.

AOI fixation duration - With the filtered data, we sum the duration of each fixation in the AOI to get the total duration.

Regressions in AOI - By definition, regressions are eye movements that go against the normal reading order [5]. They are described as backtracking fixations. Given a time stamp and location in the x and y axes in the data, we identify fixations that are an outcome of participants performing regressions. Taking the total number of regressions allows us to use this metric.

We group by AOC category in each dataframe, which results in a total count for each participant for the given AOC category. An example data frame on the total fixation count is shown in Table 4.2.

Table 4.2: Total fixation counts for all participants by AOC category.
Note: Not all participants are displayed to keep the table readable.

AOC Category	003	004	005	007	008	009	010	011	012	013	014
Conditional operators	9	132	61	68	80	217	119	200	52	132	132
Variable reassignment	31	48	39	79	39	88	47	29	92	84	37
Type conversion	6	39	42	70	22	0	45	23	31	31	0
Pre/postindecrement	76	54	126	136	51	110	183	199	75	99	115
Scope indentation	144	136	197	175	87	361	329	348	123	194	238

By utilizing the Shapiro-Wilk test [34], we confirm that the distribution of the data is not normal, meaning that we need to conduct the Kruskal-Wallis test [24] as a non-parametric version of One-Way Anova to look for significant differences in the group. These statistical tests are available in the `Python` library Scipy *v*1.13.1 [42].

4.2.3 Analysis Procedure for RQ2: Programmer Expertise

> **RQ2:** Do novices and experts experience different levels of confusion when encountering the same atom?

Having access to the demographic data of the participants, we separate the participants into three groups based on years of experience in programming: novice, intermediate, and expert.

We conduct a separate analysis on these groups for each of the five AOC categories. We store the data for novices, intermediates, and experts in separate dataframes for each category. An example of such a dataframe is shown in Table 4.3.

Having the data structured in this way allows us to run statistical tests to check for normality, equality of variances, and significant differences between the experience groups on each metric.

4.2.4 Analysis Procedure for RQ3: Prediction of Atom of Confusion

> **RQ3:** How accurately do confusion measurement metrics predict the likelihood of an atom being present in a code snippet?

Table 4.3: Data metrics for participants with intermediate experience on snippets in the Confusing Boolean with Arithmetic category.

Data Metrics Intermediate	005	012	013	016	017	024	106
Total Count Intermediate	61.00	52.00	132.00	37.00	92.00	100.00	67.00
Total Count in AOI Intermediate	53.00	39.00	82.00	25.00	77.00	74.00	35.00
Total Duration in AOI Intermediate	23.67	28.26	47.56	9.85	35.67	43.08	22.19
Total Duration Intermediate	27.14	33.24	67.88	14.57	41.24	56.65	39.17
Total Regressions Intermediate	22.00	10.00	32.00	7.00	28.00	32.00	11.00

Predicting whether a snippet contains an atom of confusion or not is a binary classification problem [15]. Therefore, we trained multiple machine learning models to find one that yields the best result.

In order to make predictions on the data, we train a machine learning model and evaluate its performance. We start by choosing metrics to use as features on which we train our machine learning model. The features that we use in our model are the following: fixation count, mean fixation duration, AOI fixation count, regressions in AOI, duration in AOI, completion time, and correctness. The variable we want to predict is the variant, which corresponds to 1 if the snippet does not contain an atom of confusion and 0 if it does. One row of these entries corresponds to the data derived from one participant on a specific snippet. The features are normalised, and the data is split into train and test data. The test data represents 30% of the whole dataset. To train and fit the models, we use `scikit-learn` *v*1.7.2 package [28]. The models that we fit are the following:

SVM with the radial basis function. The soft margin SVM features two hyperparameters: C and γ. The C parameter controls how much the model is penalized when a misclassification occurs. The smaller C is, the more points are misclassified, hence leading to an underfit, and the bigger C is, the fewer points get misclassified with a potential to overfit [14]. γ controls how much an observation influences the decision boundary. As γ increases, the decision boundary becomes more non-linear since every observation influences the boundary, and as it decreases, the fit becomes more linear.

To find the optimal values for the mentioned hyperparameters, we followed a nested cross-validation approach. We cross-validate over possible values for C and γ to pick the best hyperparameters for our data. The set of possible values for $\gamma = [100, 10, 1, 0.1, 0.01, 0.001]$ and $C = [0.1, 1, 10, 100, 1000, 10000]$. For our model, the optimal values for the hyperparameters chosen by cross-validation are $C = 10$ and $\gamma = 0.1$.

Next, we perform cross-validation on 10 folds of data, training the model on 9 folds of data, and testing on the left-out fold.

XGBoost is an ensemble method that combines multiple weak models to form a stronger model [8]. XGBoost takes numerous parameters that are required to train the model. We shortly define each parameter:

- **Max depth**: Indicates the maximum depth of the tree. Increasing this value yields a more complex model, which is more likely to overfit. For our model, we picked $max\_depth = 8$.
- **Learning rate**: Determines the rate at which the boosting algorithm learns from each iteration. Lower value indicates slower learning, as it scales down the contribution of each tree in the ensemble, preventing overfitting. For our model, we picked $learning\_rate = 0.003$.
- **Alpha**: L1 regularization parameter on weights. Increasing α will make the model more conservative. For our model, we picked $\alpha = 7$.
- **N estimators**: Specifies the number of trees to be built in the ensemble. Adding too many trees to the ensemble can lead to overfitting. For our model, we picked $n\_estimators = 1000$.
- **Gamma**: Controls the minimum amount of loss reduction required to make a further split on a leaf node. A lower value means XGBoost will stop earlier, but may not find the best solution. A higher value means the model continues training longer, potentially finding better solutions, but might lead to overfitting. For our model, we picked $\gamma = 0.1$.

We performed a randomized cross-validation search to find the parameters that yield the best results for our data. XGBoost returns the best estimator, which we use to fit the data and evaluate the performance of the model.

Random Forest Random forest is an ensemble method that combines the predictions of multiple decision trees to produce a more accurate and robust result. We set the following parameters for our model:

- **Max depth**: Indicates the maximum depth a tree is allowed to reach. For our model, we picked $max\_depth = None$. None means the nodes are expanded until all leaves are pure or until all leaves contain less than two samples.
- **Criterion**: The function to measure the quality of a split. For our model, we picked the *gini* criterion.
- **Max features**: The number of features to consider when looking for the best split. For our model we picked $max\_depth = \sqrt{|features|}$.

To train the random forest classifier, we took a similar approach as the SVM. We performed a nested cross-validation; the inner cross-validation is used to pick the best values for the parameters, whereas the outer cross-validation is used to train the model on 9 folds of data and test it on 1 fold. This yields 10 accuracy scores, of which we took the average. We used the Gini criterion as the function that measures the quality of a split; the square root of the number of features was used to consider when looking for the best split, and a depth of 8 was chosen as the maximum depth for the trained trees.

5 Results

In this chapter, we present the results and findings from our data analysis.

5.1 Results for RQ1: Categories of Atom of Confusion

Previous research on AOC studies the impact of atoms on the program comprehension of participants, accounting for the atoms uniformly, and analyzing their effect as a whole. It is reasonable to believe that some atoms require more attention and a higher concentration to understand.

This question aims to identify which atoms require a higher level of comprehension from participants to understand the construct. We present statistical results for each AOC category, analyzing eye tracking metrics such as the total fixation count, the fixation count in the AOI, the duration spent fixating in the AOI, and the total regressions in the AOI.

> **RQ1:** Do certain atoms cause more confusion than others, and if so, which ones?

We computed descriptive and inferential statistics on the relevant eye-tracking metrics to test for significant differences between the AOC categories.

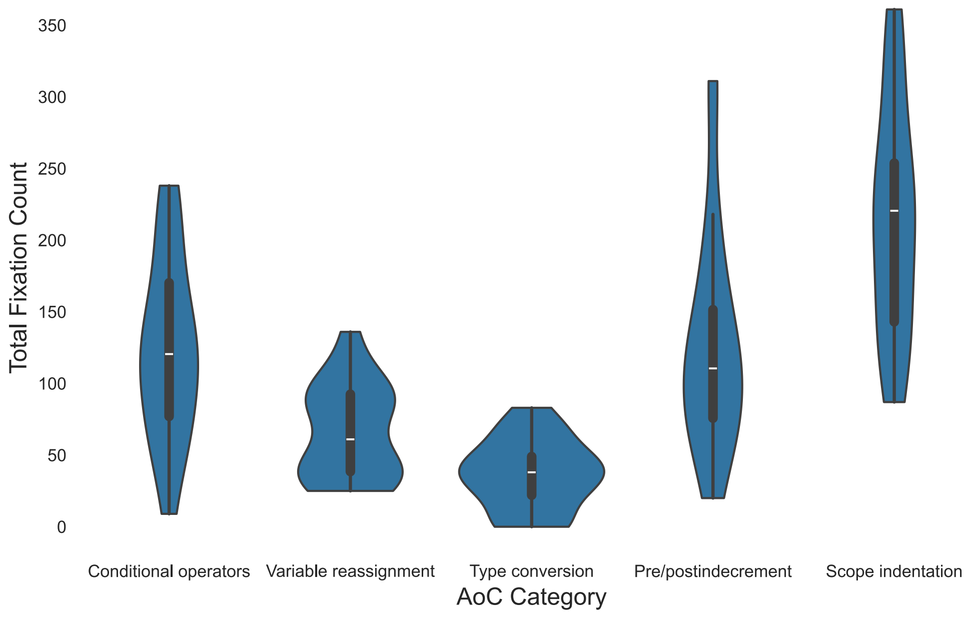

Figure 5.1: Violin plot depicting the distribution of the fixation count for each category

Total Fixation Count Figure 5.1 shows the distribution of the total fixation count for each AOC category.

The *scope indentation* category has the widest spread of the total fixations and reaches the highest values among the other categories. This is a reflection of the test results obtained from Dunn's test, where this category differs consistently from the others. Furthermore, the *variable reassignment* and *type conversion* categories have a narrower spread, with the fixation values being closer and more consistent with each other.

We then performed the Kruskal-Wallis test on the Total Fixation Count metrics across the five AOC categories, reporting a p-value of < 0.001, which indicates a high statistical significance. The Kruskal-Wallis test reports if there is a statistically significant difference between groups, but it does not describe which groups significantly differ from each other. To reveal which groups significantly differ from each other, we conduct a post-hoc Dunn's test with Bonferroni adjustment.

Results show that the *conditional operators* category differs significantly in fixation counts from three other categories, i.e *Variable reassignment*, *type conversion*, and *scope indentation*.

Table 5.1: Significant differences between pairs from Dunn's post-hoc test on fixation counts by AOC category

AOC Category 1	AOC Category 2	p-value
Conditional operators	Variable reassignment	0.038
Conditional operators	Type conversion	<0.001
Conditional operators	Scope Indentation	0.042
Variable reassignment	Pre/post in/decrement	0.043
Variable reassignment	Scope Indentation	<0.001
Type conversion	Pre/post in/decrement	<0.001
Type conversion	Scope Indentation	<0.001
Pre/post in/decrement	Scope Indentation	0.036

Moreover, *variable reassignment* significantly differs from *pre/post in/decrement* and *scope indentation*, with the same being true for *type conversion* as well. The important thing to note here is that *scope indentation* differs consistently from the other categories. Table 5.1 shows the p-values associated with statistically significant differences between the groups.

Table 5.2: Significant differences between pairs from Dunn's post-hoc test on fixation counts in AOI by AOC category

AOC Category 1	AOC Category 2	p-value
Conditional operators	Variable reassignment	<0.001
Conditional operators	Type conversion	<0.001
Conditional operators	Scope indentation	<0.001
Variable reassignment	Pre/post in/decrement	0.015
Type conversion	Pre/post in/decrement	<0.001
Pre/post in/decrement	Scope indentation	0.021

AOI Fixation count Following the same approach, we performed statistical tests on the total fixation count in the AOI for each AOC category. Table 5.2 reflects the results of the test.

Conditional operators differ significantly in terms of AOI fixation count from *variable reassignment*, *type conversion*, and *scope indentation*. *Pre/pos in/decrement* differs significantly from *variable reassignment*, *type conversion*, and *scope indentation*.

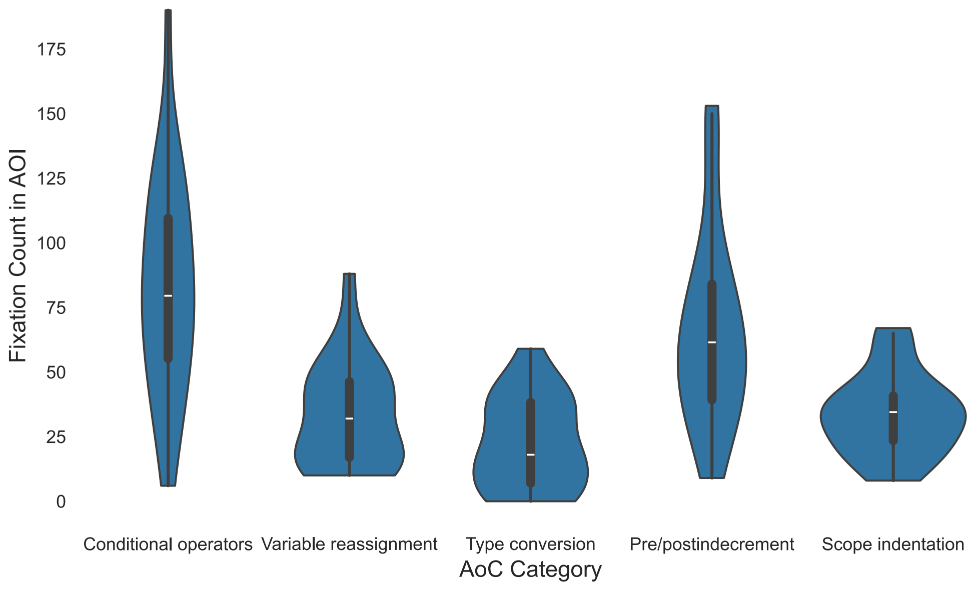

Figure 5.2: Violin plot depicting the distribution of total fixation count in AOI per AOC category

Depicting the distribution of the data in a violin plot yields some interesting results. Figure 5.2 shows the distribution of the data.

Conditional operators category has by far the widest spread of the data. Contrasting with the previous plot, when taking into account only fixation in AOI, the *scope indentation* category has a very narrow spread, with the values centering closer around the median.

Table 5.3: Significant pairs from Dunn's post-hoc test on total fixation duration in AOI by AOC category

AOC Category 1	AOC Category 2	p-value
Conditional operators	Variable reassignment	0.002
Conditional operators	Type conversion	< 0.001
Conditional operators	Scope indentation	< 0.001
Variable reassignment	Pre/post in/decrement	0.008
Type conversion	Pre/post in/decrement	0.003
Pre/post in/decrement	Scope indentation	< 0.001

AOI total duration Similar to the previous metrics, we performed the Kruskal-Wallis Test and Dunn's post hoc test to find significant differences between the AOC categories. Table 5.3 shows the results of the test. The category of *conditional operators* differs significantly from the other three categories. This is a consistent difference when taking into account the significance of the previous metrics.

Moreover, *confusing Variables* and *type conversion* differ significantly from *pre/post in/decrement*. *Pre/post in/decrement* differs significantly from *scope indentation*.

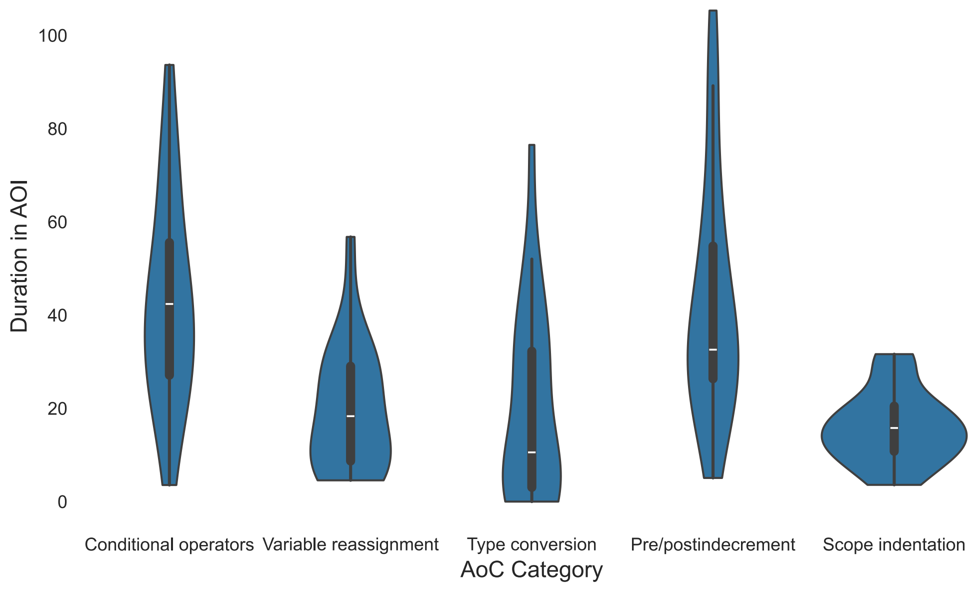

Figure 5.3: Distribution of AOI total fixation duration by AOC category.

Analyzing the plot in Figure 5.3, we notice that *conditional operators* and *pre/pos in/decrement* have the widest spread of the data points and reach the highest values. The other categories have a narrower spread, with *scope indentation* having the highest concentration of data in a small region.

Table 5.4: Significant pairs from Dunn's post-hoc test on total regressions in AOI by AOC category

AoC Category 1	AoC Category 2	p-value
Conditional operators	Variable reassignment	< 0.001
Conditional operators	Type conversion	< 0.001
Type conversion	Pre/post in/decrement	0.008
Type conversion	Scope indentation	0.001

Regressions in AOI Performing the Dunn's post hoc test on the regression total count of the categories, we got the following results:

The *conditional operators* category differed significantly in regression counts from *variable reassignment* and *type conversion* categories. Additionally, the *type conversion* category differed significantly in regression counts from *pre/post in/decrement* and *scope indentation* categories. These results are depicted in Table 5.4.

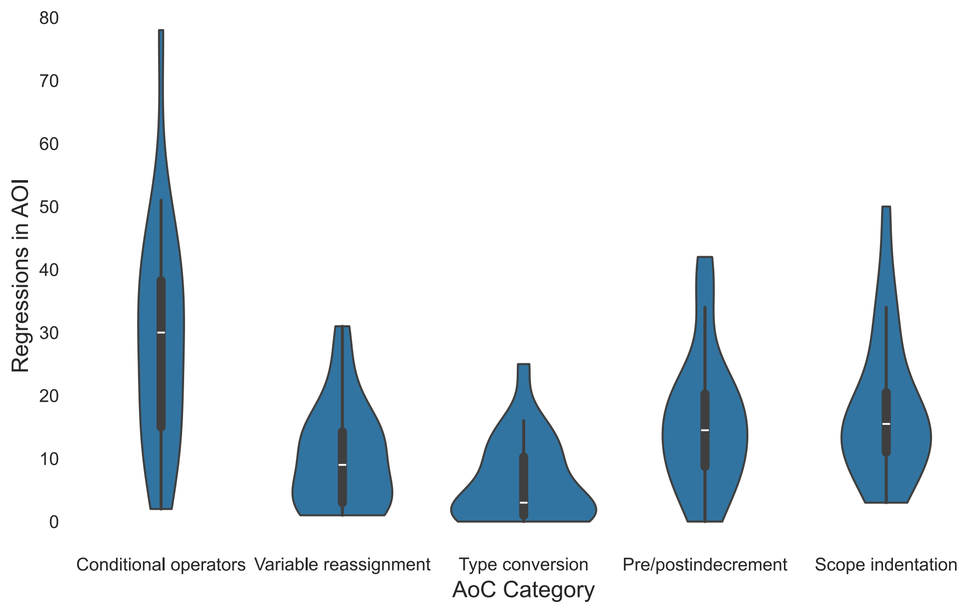

Figure 5.4: Distribution of AOI total regressions by AOC category.

Comparing the distribution plots of the regressions in AOI for each category, there are a few things to note. The distribution of the observations for the *conditional operators* category has the widest spread and values reaching the highest compared to the other categories. The other categories have a wider shape around the median, indicating that most observations are centered around this area. The *type conversion* category has the lowest median and values out of all categories, indicating little regression. The violin plot Figure 5.4 reflects these results.

5.2 Results for RQ2: Programmer Expertise

Experience plays an important role in the ability to comprehend and understand code [29, 30]. With experience, we get more familiar with the language and with its constructs. An individual who is not familiar with the language might express confusion even with code that is well-written and not confusing.

With this research question, we answer the question of whether experience plays a role in the comprehension of AOC. We analyze each of the five categories of AOC on four eye tracking metrics.

RQ2: Do novices and experts experience different levels of confusion when encountering the same atom?

5.2.1 Conditional Operators

Total fixation count Looking at the total fixation counts of each experience level, novices had the highest count with a total of 1225 fixations across all snippets in the category, and an average of 24 fixations. Interestingly, experts have the second highest count of fixations at a total of 1189 and an average of 17.7 fixations. The intermediate level has the lowest total count of fixations at 541 but an average closer to experts at 15 fixations. Thus, there is a substantial difference in total fixations and average between novices and intermediate participants.

Table 5.5: Dunn's Test result p-values between pairs of the experience levels in total fixation count.

Category 1	Category 2	p-value
Expert	Intermediate	0.636
Expert	Novice	0.199
Intermediate	Novice	0.018

These results are supported by our statistical tests. The Kruskal-Wallis test results in a p-value of 0.019, indicating statistically significant differences. The results of the subsequent Dunn's test revealed no statistically significant differences between expert-intermediate participants and expert-novice participants ($p = 0.636$ and $p = 0.199$, respectively). Whereas, between intermediate-novice groups, there is a statistically significant difference in fixation counts ($p = 0.018$).

Table 5.5 displays the associated p-values between each experience group.

AOI fixation count For novices, 759 fixations across all snippets fell in the AOI with an average count of 14.8 fixations per participant per snippet; Intermediate participants have a total of 385 fixations in the AOI and with a lower average of 10.7, meanwhile experts have a total of 852 fixations in the AOI with an average of 12.7 fixations per participant.

Even though there seems to be a big difference in total fixations between intermediate participants and the other levels of experience, these differences are not statistically significant, as shown by the Kruskal-Wallis test result with a p-value $p = 0.11$.

Total fixation duration in AOI In total, novices spent 413 seconds fixating on the AOI. On average, the total duration of the fixations is 8 seconds. Intermediate participants fixated for about half the time of novices at 210 seconds and with a lower average of 5.8 seconds. For experts, the total time spent fixating in the AOI is closer to that of novices at 398 seconds, but an average similar to intermediate participants at 5.9 seconds.

These results are not statistically significant as evaluated by Dunn's post hoc test (see Table 5.6)

Table 5.6: Dunn's Test result p-values between pairs of the experience levels in total AOI duration.

Category 1	Category 2	p-value
Expert	Intermediate	1.000
Expert	Novice	0.101
Intermediate	Novice	0.091

Total regression count Examining the regression totals in the AOI, we observe similar values across the three groups.

Novices have a total of 262 regressions with an average of 5.4 per participant. Intermediate participants have a total of 142 regressions in the AOI, but with an average of 4.5. Experts reached a total of 287 regressions with an average of 4.4 regressions.

These resulting values are close between the groups, and performing the Kruskal-Wallis test resulted in a p-value of $p = 0.489$, which is statistically insignificant.

Summary For snippets containing atoms of the *conditional operators*, the level of experience does not influence the observed confusion, since we largely find no effect regarding our eye tracking metrics.

While there is a statistically significant difference in the total count of fixations between novice-intermediate participants, the other metrics yield no statistical significance. This especially includes the AOI metrics such as total count, duration, and regressions in AOI.

5.2.2 Pre/Post In/Decrement

Total fixation count For snippets in the *pre/post in/decrement* AOC cateogry we observe the following results: Novices fixated on the snippet for a total of 1269 fixations with an average of 19.2 fixations; Intermediate participants fixated significantly less, having a total of 691 fixations and an average of 11.9 fixations; experts have 1070 total count of fixations with an average of 14 fixations. We observe a statistically significant difference between novices and intermediate participants as there is a 54% decrease in fixation counts. Furthermore, even though not as strong, we see a difference between novices and experts in total count aswell.

The results of the test between experts and novices found a p-value of $p = 0.022$, indicating that there is a statistically significant difference in fixation counts between these groups. Moreover, there is such a significant difference between intermediate and novice participants with a p-value of < 0.001.

Table 5.7: Dunn's Test result p-values between pairs of the experience levels in total fixation count.

Category 1	Category 2	p-value
Expert	Intermediate	0.404
Expert	Novice	0.022
Intermediate	Novice	<0.001

AOI fixation count Novices displayed the highest result, reaching a total of 652 fixations in the AOI, with a mean count of 9.8. Similar to the previous metric, intermediate participants have the lowest total AOI fixation count at 339 fixations with an average of 5.8 fixations. Finally, experts lie between the two groups with 584 total AOI fixations and 7.6 average.

These trends suggest that novices and experts analyse and focus on the AOI more often than intermediate participants.

The statistical results from performing Dunn's test show that these results are not random(Table 5.8).

There is a statistically significant difference between novices and intermediate participants($p < 0,001$), indicating that novices consistently focus more on the AOI than intermediate participants. Although not as strong, there is a statistically significant difference between experts and intermediate participants with a p-value of 0.01.

Table 5.8: Dunn's Test result p-values between pairs of the experience levels in total AOI fixation count.

Category 1	Category 2	p-value
Expert	Intermediate	0.016
Expert	Novice	0.106
Intermediate	Novice	< 0.001

Total fixation duration in AOI Observing the time spent fixating on the AOI part of the snippet, novices once again, had the highest total duration at 438.7 seconds with a mean fixation of 6.6 seconds. Experts show the second-highest total fixation duration in the AOI at 363.2 seconds and an average of 4.7 seconds per participant per snippet. Lastly, with the lowest total duration, intermediate participants had a total of 209.6 seconds and an average of 3.6 seconds. Intermediate participants spent the least amount of time fixating on the AOI, similarly to the previous metrics.

The statistical results Table 5.9 give us insights into the pairwise differences between the groups.

We discovered a statistically significant difference in total fixation duration between novice and intermediate participants ($p < 0.001$). Moreover, we found a significant difference between novices and experts ($p = 0.006$), while the difference between experts and intermediate participants is insignificant($p = 0.3$).

These results suggest a different behavior in how long it takes novices to perceive and understand confusing constructs compared to experts and intermediate participants.

Table 5.9: Dunn's Test result p-values between pairs of the experience categories in total duration fixating in AOI.

Category 1	Category 2	p-value
Expert	Intermediate	0.326
Expert	Novice	0.006
Intermediate	Novice	< 0.001

Total regression count The results of the total regression count metric show statistically significant pairwise differences between the groups. The highest number of total regressions in the AOI was achieved by novices, with a total of 161 regressions into the AOI with a mean of 2.8 regressions per participant per snippet. Experts are second, with a total of 152 regressions and a lower average of 2.6 regressions. Intermediate participants had the lowest total of regressions in the AOI at 67 regressions and an average of 1.97 regressions in the AOI. We discover that experts and novices have a similar count and average of regressions, while the metrics for the intermediate participants are noticeably lower.

Looking at the statistical results of the pairwise differences between the categories, intermediate participants differ significantly from novices ($p = 0.023$) and experts ($p = 0.03$) in regression counts in the AOI. Meanwhile, no statistically significant difference was found between experts and novices. This lack of difference is also noted when comparing the totals and averages of these two groups.

Table 5.10: Dunn's Test result p-values between pairs of the experience categories in total regression count.

Category 1	Category 2	p-value
Expert	Intermediate	0.030
Expert	Novice	1.000
Intermediate	Novice	0.023

Summary For confusing snippets in the *post/pre in/decrement* category, we notice a few things. Across the four eye tracking metrics that we analysed, in two of them (Total fixation count and total fixation duration in AOI), we found a statistically significant difference between novices and the other two levels of experience, where novices consistently produced the highest values and averages, indicating a higher cognitive load when processing these code snippets. In the other two metrics (AOI fixation count and regressions in AOI), intermediate participants differed significantly from the other two levels of experience. They achieved the lowest values out of the three groups.

5.2.3 Scope Indentation

The statistical analysis for snippets in the *scope indentation* AOC category found no statistically significant results for any of the metrics except **total regression count**; however, we discuss the metrics below:

Total fixation count For the expert category, we recorded the highest total fixation count at 1983 fixations, but they achieved the lowest average per participant per snippet at 24.2 fixations. Second is the novice category with 1891 total fixations, but the highest average at 30.5 fixations. Similar to previous snippet categories, intermediate participants achieved the lowest total fixation count at 1239 fixations, but with an average closer to experts at 25.2. The results of the Kruskal-Wallis test found no statistically significant difference between the groups ($p = 0.11$)

AOI fixation count Taking into consideration only the fixations falling in the AOI, experts had the highest count of fixations at 354, with an average of 4.6 fixations per participant per snippet. Novices followed in second at 290 fixations with an average of 4.7 fixations. Finally, intermediate participants had the lowest AOI total and average at 165 and 3.6, respectively.

Performing the Kruskal-Wallis test, we found no statistically significant differences between any of the three groups.

Total fixation duration in AOI Experts resulted in the highest total duration spent fixating in the AOI at 157.8 seconds, with an average of 2.1 seconds. Novices fixated on the AOI for a total of 138 seconds and an average of 2.2 seconds per participant per snippet. Finally, intermediate participants spent a total of 85 seconds fixating on the AOI with an average of 1.8 seconds.

Similarly to the previous metrics, we found no statistically significant difference between the three groups for this metric ($p = 0.6$).

Total regression count When it comes to regressions in the AOI, experts resulted in the highest total at 203 regressions and an average of 3 regressions per participant per snippet. Novices had a total of 143 regressions and an average of 2.6 regressions. Finally, intermediate participants had a total of 81 regressions and an average of 2 regressions.

Performing Dunn's post hoc test, we found a statistically significant difference only between expert and intermediate participants with a p-value of $p = 0.03$ (see Table 5.11)

Table 5.11: Dunn's Test result p-values between pairs of the experience categories in total regression count.

Category 1	Category 2	p-value
Expert	Intermediate	0.031
Expert	Novice	1.000
Intermediate	Novice	0.205

summary Although we found a statistically significant difference in the total regression count between intermediate and expert participants, this metric alone does not allow us to draw any conclusion or make any implications about whether the same snippet category causes a different level of confusion between participants with different levels of experience.

5.2.4 Type Conversion

Total fixation count Intermediate participants scored the lowest total fixation count across all snippets in the category at 212 fixations with an average of 11.7 fixations per participant per snippet. Novices had a higher total count of fixations at 306 fixations with an average of 15.3 fixations per participant per snippet. Experts had the highest total count of the 3 experience levels at 351 fixation total. The resulting average per participant per snippet was evaluated at 14.6 fixations.

The results of the Kruskal-Wallis test discovered no statistically significant difference between the total fixation counts between any of the experience groups ($p = 0.4$)

AOI fixation count Observing the total fixations in the AOI for all snippets in the category, novices achieved a total of 166 fixations with an average of 8.3 fixations per snippet. Experts had the highest total count at 270 fixations. Finally, intermediate participants had the lowest total at 98 fixations and a low average of 5.4 fixations per snippet.

There is a notable difference in AOI total fixations between experts and intermediate participants. This difference is discovered to be statistically significant by the statistical results of Dunn's test with a p-value of 0.006 (see Table 5.12).

Table 5.12: Dunn's Test result p-values between pairs of the experience levels in total AOI fixation count.

Category 1	Category 2	p-value
Expert	Intermediate	0.006
Expert	Novice	0.347
Intermediate	Novice	0.391

Total fixation duration in AOI Intermediate participants fixated on the AOI for a total of 60.5 seconds across all snippets, and on average, they spent 3.3 seconds per snippet fixating in the AOI. Novices fixated on the snippets' AOI for a total of 134.7 seconds, while experts had the highest total duration of 285.2 seconds and an average of 11.8 seconds per snippet.

The results of the statistical test follow the same trend as the previous metrics. There is a notable difference in the total duration spent fixating in the AOI between experts and intermediate participants. The associated p-value to this difference is < 0.001; meanwhile, we found no statistically significant difference between intermediate-novice and expert-novice pairs (see Table 5.13).

Table 5.13: Dunn's Test result p-values between pairs of the experience levels in total AOI duration.

Category 1	Category 2	p-value
Expert	Intermediate	< 0.001
Expert	Novice	0.277
Intermediate	Novice	0.095

Total regression count Intermediate participants had the lowest count of regressions in the AOI across all snippets, with a total of 19 regressions and an average of 1.7 regressions per snippet. The second-highest value was achieved by novices with a total of 39 regressions across all snippets and an average of 2.8 regressions per snippet. Finally, novices had the highest total count of regressions in the AOI, with a total of 91 regressions and an average of 4.5 regressions per snippet.

The results of Dunn's post hoc test demonstrate a statistically significant difference between experts and intermediate participants, with a p-value of $p = 0.002$. No such significance was found between any other experience level pairs.

Summary There is a noticeable difference in the eye-tracking data between experts and intermediate participants. Intermediate participants consistently scored the lowest values in all of the above metrics, but the significant results that we found concerned mainly the AOI, that is, the atom of confusion.

Taking into account solely the eye-tracking data, we state that expert participants focused longer on the AOI, as indicated by the long fixation duration times.

Based on the regression count, they backtracked to the AOI much more often than intermediate participants. Experts gave the wrong answer 23 out of 27 times, while intermediate participants gave the wrong answer 20 out of 21 times.

These metrics suggest that experts experience a higher cognitive load when comprehending the AOC compared to intermediate participants.

5.2.5 Variable Reassignment

We found no statistically significant differences between pairwise experience levels for snippets in the *variable reassignment* category. Nevertheless, we provide descriptive statistics regarding each eye-tracking metric below:

Total fixation count We recorded 525 total fixations across all snippets in the category for participants in the novice level, with an average of 12.5 fixations per snippet. Intermediate participants had a total of 480 fixations with an average of 11.1 fixations per snippets and experts resulted in a total of 579 fixations across all snippets and an average of 11.5 fixaions per snippet.

Performing the Kruskal-Wallis test, we found no statistically significant difference between any of the experience levels, resulting in a p-value of $p = 0.8$

AOI fixation count Taking into account only fixations in the AOI, novice participants had a total of 263 fixations and 6.2 fixations on average per snippet. Intermediate participants had a total of 212 fixations across all snippets, with an average of 4.9 fixations per snippet. Lastly, experts resulted in 326 total fixations and an average of 6.5 fixations per snippet.

Performing the Kruskal-Wallis test, we found no statistically significant difference between any of the experience levels, resulting in a p-value of $p = 0.2$

Total fixation duration in AOI Novices fixated for a total of 141.6 seconds across all snippets in the category, with an average fixation time of 3.4 seconds per snippet. Intermediate participants fixated for a total of 134.9 seconds and averaged 3.1 seconds per snippet. Expert participants had a total fixation duration of 196.7 seconds and an average of 3.9 seconds per snippet.

Performing the Kruskal-Wallis test, we found no statistically significant difference between any of the experience levels, resulting in a p-value of $p = 0.5$

Total regression count For the final eye-tracking metric, we found a total of 96 regressions across all snippets for novices with an average of 3.3 regressions per snippet. Intermediate participants had a total of 56 regressions with an average of 2 regressions per snippet, and experts had a total of 93 regressions across all snippets and an average of 2.8 regressions per snippet.

Performing the Kruskal-Wallis test, we found no statistically significant difference between any of the experience levels, resulting in a p-value of $p = 0.5$

Summary We found no statistically significant difference between the experience levels when comprehending snippets of the variable reassignment AOC category. These results indicate that there is no effect on the observed confusion of the participants depending on their experience level.

5.3 Results for RQ3: Prediction of Atom of Confusion

A previous study found that AOC cause an increase in gaze transitions and time of completion [10]. Such effects in the eye-tracking and behavior data have the potential to reveal a systematic pattern in the data, which can allow us to determine whether or not an atom is present in the code.

In order to identify such patterns, we train a machine learning model on our existing dataset. We present the results of the various machine learning models we trained and the evaluated performance for each model.

> **RQ3:** How accurately do confusion measurement metrics predict the likelihood of an atom being present in a code snippet?

The goal of our machine learning models is to predict the variants of the snippets. The variant of a snippet is 1 if the snippet does not contain an atom of confusion and 0 otherwise.

We train the model on 7 features[1] and we split the data into train and test data. The training set makes up 70% of the data, and the test set makes up 30% of the data. We normalized the data before using it to train the models.

We evaluate the models on 451 data-points, where 230 data-points correspond to snippets that do not contain an atom of confusion, and the rest of 221 data points contain an atom of confusion. The evaluated performance on the hold-out set is shown below Table 5.14:

Table 5.14: Evaluated performance of the trained models

Model	Accuracy	Recall	Precision	F1 score
Soft margin SVM	0.6408	0.8788	0.6024	0.7148
XGBoost	0.6319	0.7532	0.6148	0.6770
Random forest	0.6275	0.6580	0.6307	0.6441

Evaluated on the hold-out set, we conclude that out of the trained models, the soft margin SVM model performed better compared to the other models in terms of accuracy, producing a correct label for 64% of the test data.

Furthermore, SVM scored the highest recall (0.8788). It indicates that this model performed the best in correctly predicting whether snippets do not contain an atom. This means that the SVM model is most sensitive to positive instances.

Moreover, SVM has the highest F1 score (0.714), implying a better balance between precision and recall.

XGBoost performed slightly worse than the SVM but better than the Random Forest, producing a correct label for 63% of the data. XGBoost elevates the value of precision (0.6148), implying that when it predicts a positive instance, it is more likely to be correct compared to the other models.

The random forest model scored similar values to XGBoost in accuracy of predictions ($\approx$ 63%) but achieved the highest precision score (0.6307). The recall score (0.6580) is much lower compared to XGBoost and SVM, making it the least sensitive to positive instances.

Overall, the Soft Margin SVM offers a higher performance in accuracy of predictions and is superior when it comes to predicting actual positive instances and yielding a more balanced performance. Taking into account these metrics, we can say that the Soft Margin SVM achieves the best performance in predicting whether a snippet contains an atom of confusion or not.

An important thing to note is that while these models do not achieve a high predicting performance due to the inconsistent nature of human comprehension, all the models achieve a better predicting performance over chance and by simply guessing whether a snippet contains an atom or not.

1 Features used to train the model: fixation count, mean fixation duration, AOI fixation count, regressions in AOI, duration in AOI, completion time, and correctness

6 Discussion

In this thesis, we explore and analyze the effects of atoms of confusion on the program comprehension of participants. We do this by utilizing eye-tracking data, which is widely used in the area of research. Such data provides insights into the behavior of participants when comprehending code or natural language.

In this chapter, we provide an interpretation of the results for each research question in order to give a thorough answer to each one of them.

6.1 RQ1: Categories of Atom of Confusion

> **RQ1:** Do certain atoms cause more confusion than others, and if so, which ones?

We analyze the five grand categories of AOC: *Conditional Operators*, *Variable Reassignment*, *Type Conversion*, *Pre/Post increment*, and *Scope Indentation* across different eye tracking metrics. The statistical analysis we performed for each eye-tracking metric revealed one common result: The *conditional operators* category and the *pre/post increment* category consistently differed the most from the other categories.

The violin plots demonstrate that these two categories most often have the widest spread, and reach the highest values and medians in the data. The spread of the values indicates inconsistency in how participants perceive the atoms.

In metrics that isolate the AOI like AOI fixation count, AOI fixation duration, and regressions, these two categories have the highest counts and values. A high fixation count and fixation duration indicate that participants spent a significant amount of time focusing on the AOI area, perhaps trying to understand what the confusing part of the snippet evaluates to.

A reason why these two categories received the most attention compared to others is due to the nature of their constructs. Taking an example of an *increment operator*, we have two possible options: pre-increment and post-increment. One increments the value before used in an expression, and the other increments the value after used in an expression. Although simple when the construct appears alone, when combined with arithmetic and/or assignment to variables, the construct becomes more complex, making it harder to hand-evaluate or comprehend [11].

When it comes to *conditional operators*, a similar problem may occur [11]. When a statement contains multiple conditional operators without the necessary parentheses, it gets too complex to understand how the operators bind, and even then, evaluating the results of each conditional statement causes confusion and longer comprehension times.

In the study of Silva da Costa et al. [9], they found that snippets with atoms of the type "operator precedence" significantly increased the number of total fixations, regressions, and fixation duration. In our study, we have referred to this category as *conditional operators*. While our study focused on finding which category of atoms causes the most confusion, Silva da Costa et al. [9] focused on analyzing the difference between clarified counterparts. Our results align with those of Silva da Costa et al.[9] in a way that *conditional operators* (operator precedence in their study) had a significantly higher number of fixations, regressions, and fixation duration.

Further research would include analyzing *conditional operators* and *pre/post in/decrement* categories further, perhaps dividing the snippets into sub-categories to find whether certain types of constructions cause an increase in the eye-tracking measures used in our study. This will help to narrow down the constructs that make participants and developers more prone to making errors.

In a real-life setting, our results indicate that educators should refrain from using and creating code that includes atoms of the *conditional operators* and *pre/post in/decrement* category. The presence of these atoms in teaching material will significantly increase the visual effort required from students to understand these atoms.

6.2 RQ2: Programmer Expertise

> **RQ2:** Do novices and experts experience different levels of confusion when encountering the same atom?

It is reasonable to believe that experience plays an important role in how fast and correctly developers comprehend code. Previous eye-tracking studies have concluded that participants with a higher level of experience can comprehend code at a much faster rate when compared to participants with less experience [29, 30].

We separated the participants in our study based on their experience level, and we analyzed the eye-tracking data to find out how atoms of confusion affect the eye behavior of the participants when encountering them.

For AOC categories, such as *conditional operators*, *scope indentation*, and *variable reassignment*, we found no significance between the experience levels in the observed confusion on metrics that isolate the atom of confusion. We found significant results for the *pre/post in/decrement* and *type conversion* categories, which we will discuss here.

It is important to note that the AOI of snippets belonging to the *type conversion* AOC category takes up the largest area of the snippet.

When encountering snippets of pre/post in/decrement category, the results show that novices consistently produced the highest values in total fixations and focused the longest in the AOI area compared to intermediate and expert participants. They achieved the highest average and total with respect to these metrics, whereas intermediate participants, although not significant, had the lowest average and lowest total out of the three levels of experience.

This result can be credited to the fact that novices, due to the lack of experience and familiarity with code, are required to focus on every keyword and statement of the snippet. They need all the pieces of the snippet to produce some mental model of what the snippet does to evaluate and understand it. This justifies the high fixation count across the snippet and the long fixation duration times.

For the type conversion category, by analyzing the eye tracking metrics, we found that intermediate participants and experts had a significant difference in the AOI fixation count and the number of regressions in the AOI. In both metrics, intermediate participants achieved the lowest total count and the lowest average.

One may expect that experts would produce the lowest values out of the other experience levels, but there is one aspect to consider. As mentioned before, novices tend to read more linearly, focusing on each line and on the surface-level syntax due to their lack of experience, familiarity, or overall knowledge about the programming language [11, 29, 30]. On the other hand, experts read code less linearly, utilizing prior knowledge and experience, and being able to scan code more selectively [29, 30], which results in lower fixation and regression counts.

Intermediate participants can be a combination of both. Having some experience and knowledge about the language allows them to skip certain parts of the code. Their knowledge might come to an end when encountering more complex constructs to which they don't pay attention, resulting in lower values for the respective eye-tracking metrics.

We found no significance in the accuracy of the answers between any of the groups. This measure does not draw any conclusion regarding the confusion caused to the participant. This was also concluded by Gopstein et al. [11], where it is stated that providing a correct answer does not imply (mis)understanding the snippet.

We used the years of experience with programming languages as our measure of experience, which might not represent the true experience and overall code comprehension skills [29]. Feature research can include measures such as the one introduced by Peitek et al. [29], where they use efficacy as a measure of programming skill level. This could yield more contrasting results between groups of different efficacy levels.

6.3 RQ3: Prediction of Atom of Confusion

> **RQ3:** How accurately do confusion measurement metrics predict the likelihood of an atom being present in a code snippet?

The aim of this research question was to determine, based on eye-tracking and behavior data, whether we can predict the presence of an atom of confusion in a code snippet. We trained multiple classification models, and we evaluated their performance on a hold-out set.

The Soft Margin SVM offers a higher predictive performance compared to XGBoost and Random Forest. It achieves a superior Accuracy and Recall score (0.6408 and 0.8788, respectively) while also offering the best balance between recall and precision.

In Figure 6.1, we display the confusion matrix of the Soft Margin SVM. The model correctly predicted the absence of an atom of confusion 203 out of 231 times, indicating that this model is very sensitive to positive instances. It correctly predicted the presence of an atom of confusion 86 times, and it has produced a false result 134 times. The model performs worse in predicting snippets that contain an atom of confusion.

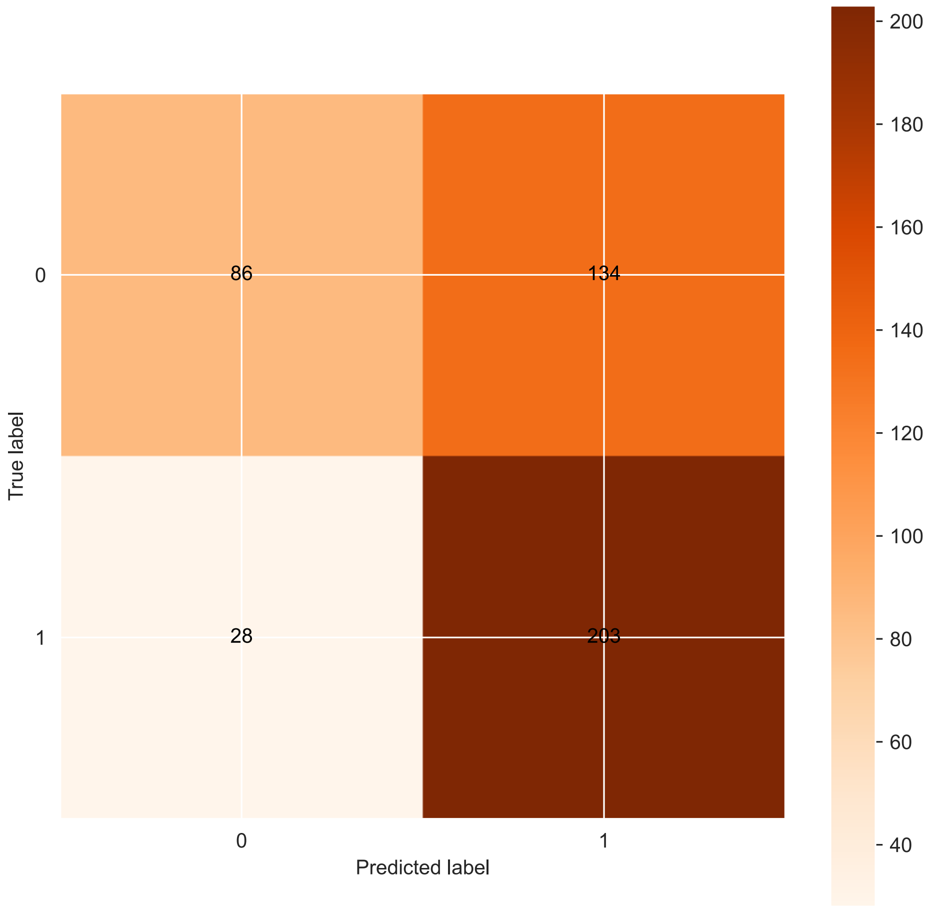

Figure 6.1: Confusion matrix of the Soft Margin SVM. 1 corresponds to a snippet that does not contain an atom and 0 otherwise.

The XGBoost model achieved a higher 111 correct classifications for snippets that did contain an AOC compared to the SVM. Predicting correct classification for snippets that do not contain an AOC is a more difficult task for XGBoost, with 174 correct predictions as compared to 203 correct predictions of the SVM. These results can be seen in Figure 6.2

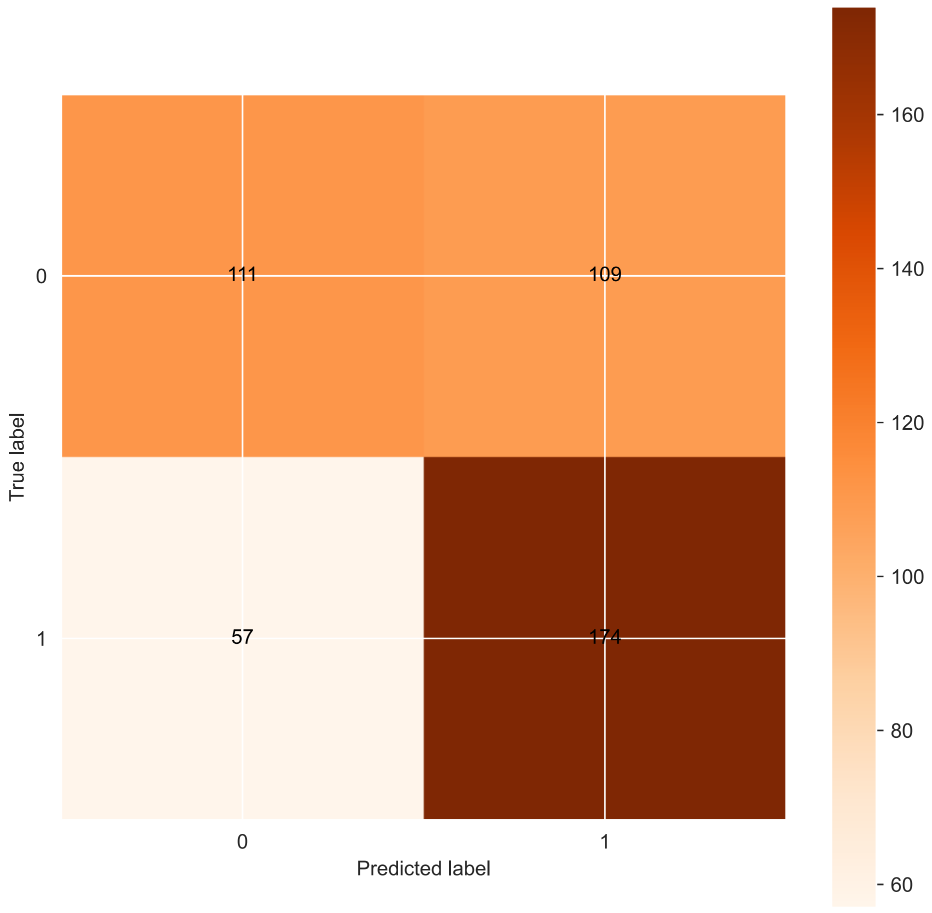

Figure 6.2: Confusion matrix of the XGBoost model. 1 corresponds to a snippet that does not contain an atom and 0 otherwise.

Finally, the Random Forest model achieved the best performance in predicting correct classifications for snippets that contained an atom of confusion, with a total of 131 out of 220 correct predictions. The model performed poorly in predicting the absence of an AOC in the snippets, with 152 correct predictions and 79 incorrect predictions Figure 6.3.

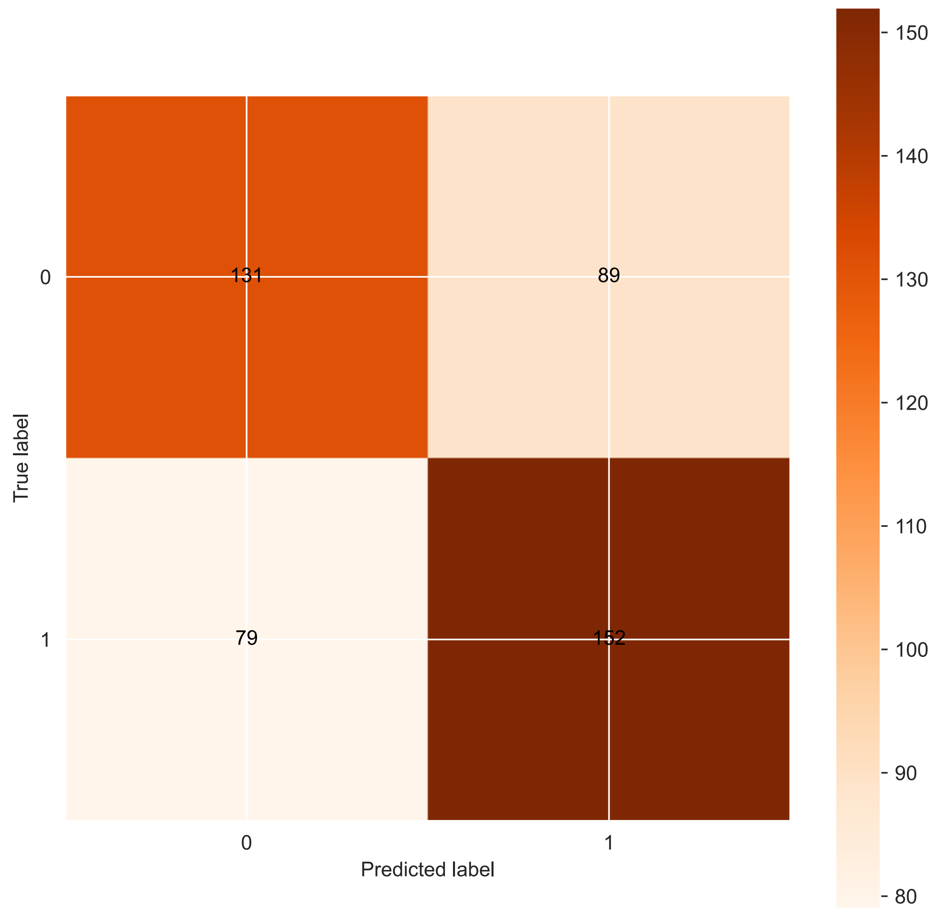

Figure 6.3: Confusion matrix of the Random Forest model. 1 corresponds to a snippet that does not contain an atom and 0 otherwise.

The evaluation metrics of the models indicate that the SVM achieves the best performance out of the three trained models. Generally, this would be the model of choice to use, but in the scope of our research question, we want to know how well we can predict the presence of the atom. Under these circumstances, the Random Forest model performs the best as it has the highest number of true negatives and the lowest false positives (i.e., correctly predicting if a snippet contains an atom, and predicting the absence of an atom when the atom is actually present).

A limiting factor to the performance of the models can be the size of the dataset [33, 41]. Models trained on small datasets tend to overfit the training data, hence producing results that do not represent the true data. To mitigate this risk of overfitting, we performed cross-validation on different values for the hyperparameters of the models. This allows us to choose the best hyperparameters for our models and data to reduce the risk of overfitting.

To this end, future work may include increasing the participant size by conducting additional experiments, which would add to the existing dataset.

The existing dataset contained a rating that the participants gave to the snippets. The rating data consists only of "hard" and "easy". In future experiments, introducing a Likert scale for the difficulty of the snippet could potentially improve the models when used as a feature to make the predictions.

Our best machine learning model, the SVM, was able to predict the presence or absence of an atom in the code with 64% accuracy. Although this accuracy is not high by most machine learning models, our model predicts better than chance. Potentially, with a larger dataset, we can train a better model with higher accuracy. In a setting where we are only concerned with predicting the presence of an atom, our Random Forest model performed the best with a precision score of 0.6307.

These results have provided meaningful insights into the effects of atoms of confusion in program comprehension. Our results serve as a basis for future work in investigating and revealing which atoms cause the most confusion and to whom.

7 Threats to Validity

In this chapter, we discuss the possible threats and issues with our results and findings of the study.

7.1 Internal Validity

The code snippets that represent the atom candidates in this study contain variables with non-descriptive names like R, $V1$, and $V2$. This naming is used because the snippets are short and isolated. Giving the variables meaningful names may introduce bias in the end result. Nevertheless, non-descriptive names reduce comprehension of the program [37], which adds a layer of noise in our data.

To categorize participants in experience levels, we used the years of experience as a measure. This might not truly represent the participants' programming skills, and specifically, programming skills in `Java`. The demographic data gathered from participants contains 1 if the participant had contact with Java and 0 otherwise. This does not reflect the participants' confidence in `Java`, which might affect the results of our analysis.

7.2 External Validity

The snippets in our dataset are constructed in the `Java` programming language. The experiments and the data gathered during the experiments originate from these snippets. This means that the research questions, hypotheses, findings, and results in this thesis cannot be generalized to other programming languages. This is due to the difference in syntax, semantics, flow of the program, and logic of writing between different programming languages. For example, in languages such as `Python`, indentation plays a role in the control flow of programs. Indentation is part of the syntax, so atoms of the category scope indentation behave in different ways depending on how the atom is indented and possibly produce errors.

The set of atoms used in this study does not objectively include all possible atoms of confusion in `Java`, nor do they include all the categories. This means other atoms like *reverse subscript* or *pointer arithmetic* are not covered in this thesis, and therefore the results do not apply to these categories.

The participants in this study are predominantly university students at Bachelor's or Master's levels. Their programming experience, problem-solving, and familiarity with code might differ significantly from those of professional developers in industry settings. The findings in this study may not be generalizable to real-world, professional developers.

Our machine learning models have been trained on eye-tracking data gathered from snippets that are constructed in Java. They might not generalize and provide valuable predictions for eye-tracking data gathered from snippets of other programming languages.

7.3 Construct Validity

Prior eye-tracking studies on atoms of confusion [9, 10, 40] measure confusion by comparing the increase in eye-tracking measures between clarified snippets and their obfuscated counterparts. This comparison allows the eye-tracking metrics to be interpreted relative to a baseline, not confused eye-behavior.

As our study focuses on comparing types of obfuscated code snippets of different categories, there is no explicit baseline that represents non-confusion. The reported eye-tracking measures depict relative differences between atoms of confusion. Therefore, the results of these metrics may not allow for a direct conclusion about how much more confusing one atom is compared to another.

8 Conclusion

In this thesis, we analyzed the effects of the atoms of confusion on the participants' program comprehension by analyzing different eye-tracking metrics. More precisely, we compared how much different categories of atoms of confusion affect the eye behavior of participants. We analyzed the effect these atoms have on participants separated by experience in coding to reveal whether atoms cause confusion only to a certain group of participants. We trained and evaluated three machine learning classification models to assess, based on eye-tracking and behavior data, whether code snippets contain an atom of confusion or not. This provides a foundation for future research to explore the capabilities of machine learning in assessing the quality of code and potentially detecting new atoms that have not been considered before.

By analyzing the total fixation count, AOI fixation count, AOI fixation duration, and regressions, we were able to compare which categories of atoms caused more confusion to participants and to which experience levels. Overall, we found that snippets of the conditional operators and pre/post in/decrement category caused the longest fixation times, the most fixations in the AOI, and the most regressions. We observed a total AOI fixation count of 1996 fixations for the conditional operators category and 1575 total fixations for the pre/post in/decrement category. This corresponds to an increase of more than double the fixations of the closest category, which is the scope indentation category, with 809 AOI fixations.

When factoring in the participants' level of experience, we found no statistically significant difference in the observed confusion for categories: conditional operators, scope indentation, and variable reassignment. Experience does not play a role in the confusion caused by these snippet categories. For snippets in the categories of pre/post in/decrement and type conversion, we found that novices produced significantly higher AOI fixation counts and regressions compared to intermediate participants. Whereas experts had significantly higher AOI fixation counts, fixation duration in AOI, and regressions in AOI compared to intermediate participants, which produced the lowest values.

Utilizing the eye-tracking and behavior data, we trained three machine learning classification models to predict the presence or absence of an atom in the snippets. Our best performing model, the SVM, achieved an accuracy score of 64%, which improves significantly over chance and guessing.

Future work shall include a deeper analysis of the subcategories of AOC. Analyzing what form of atoms for each category causes more confusion will help in narrowing down and finding which atoms are more severe in causing confusion and hindering program comprehension.

Including other experience measures, such as efficacy [29], confidence in the knowledge of the programming language will allow for better separation between participants in experience levels. This separation might yield even better insights into how participants of different experience levels behave when encountering AOC.

Finally, future work shall include the gathering of more eye-tracking data by expanding the participant count. Having a greater data set might help improve the accuracy of the machine learning models. This will allow us to identify confusing code and perhaps atoms that have not been considered yet to be confusing.

Statement on the Usage of Generative Digital Assistants

For this thesis, the following generative digital assistants have been used: We have used ChatGPT as a writing assistant. We have used it only to the extent of improving language clarity and, on some occasions, for paraphrasing. We have never prompted ChatGPT to generate full sections or paragraphs. All ideas, methods, and research questions are our own.

We used Microsoft Copilot for code generation. It never generated full code files or sections, but rather short lines of code, as a way to speed up the coding process.

We are aware of the potential dangers of using these tools and have used them sensibly with caution and with critical thinking.

Bibliography

[1] Magdalena Andrzejewska and Agnieszka Skawińska. “Examining students’ intrinsic cognitive load during program comprehension–An eye tracking approach.” In: *International Conference on Artificial Intelligence in Education*. Springer. 2020, pp. 25–30.

[2] Roman Bednarik. “Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations.” In: *International Journal of Human-Computer Studies* 70.2 (2012), pp. 143–155.

[3] Annabelle Bergum, Anna-Maria Maurer, Norman Peitek, Regine Bader, Axel Mecklinger, Vera Demberg, Janet Siegmund, and Sven Apel. “Unexpected but informative: What fixation-related potentials tell us about the processing of ambiguous program code.” In: *arXiv preprint arXiv:2412.10099* (2024).

[4] Mike Bland. “Finding more than one worm in the apple.” In: *Commun. ACM* 57.7 (July 2014), 58–64. ISSN: 0001-0782.

[5] Robert Booth and Ulrich Weger. “The function of regressions in reading: Backward eye movements allow rereading.” In: *Memory cognition* 41 (Aug. 2012).

[6] Teresa Busjahn, Carsten Schulte, and Andreas Busjahn. “Analysis of code reading to gain more insight in program comprehension.” In: *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*. Koli Calling ’11. Koli, Finland: Association for Computing Machinery, 2011, 1–9. ISBN: 9781450310529.

[7] Siyuan Chen, Julien Epps, Natalie Ruiz, and Fang Chen. “Eye activity as a measure of human mental effort in HCI.” In: *Proceedings of the 16th international conference on Intelligent user interfaces*. 2011, pp. 315–318.

[8] Tianqi Chen. “XGBoost: A Scalable Tree Boosting System.” In: *Cornell University* (2016).

[9] José Aldo Silva da Costa, Rohit Gheyi, Fernando Castor, Pablo Roberto Fernandes de Oliveira, Márcio Ribeiro, and Baldoino Fonseca. “Seeing confusion through a new lens: on the impact of atoms of confusion on novices’ code comprehension.” In: *Empirical software engineering* 28.4 (2023), p. 81.

[10] Benedito De Oliveira, Márcio Ribeiro, José Aldo Silva Da Costa, Rohit Gheyi, Guilherme Amaral, Rafael de Mello, Anderson Oliveira, Alessandro Garcia, Rodrigo Bonifácio, and Baldoino Fonseca. “Atoms of confusion: The eyes do not lie.” In: *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. 2020, pp. 243–252.

[11] Dan Gopstein, Anne-Laure Fayard, Sven Apel, and Justin Cappos. "Thinking aloud about confusing code: A qualitative investigation of program comprehension and atoms of confusion." In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 605–616.

[12] Dan Gopstein, Jake Iannacone, Yu Yan, Lois DeLong, Yanyan Zhuang, Martin K-C Yeh, and Justin Cappos. "Understanding misunderstandings in source code." In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 2017, pp. 129–139.

[13] Dan Gopstein, Hongwei Henry Zhou, Phyllis Frankl, and Justin Cappos. "Prevalence of Confusing Code in Sotware Projects." In: (2018).

[14] Trevor Hastie. *The elements of statistical learning: data mining, inference, and prediction*. 2009.

[15] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. In: *An introduction to statistical learning*. New York, NY, USA: Springer, 2013. ISBN: 9781461471370.

[16] Ahmad Jbara and Dror G Feitelson. "How programmers read regular code: a controlled experiment using eye tracking." In: *Empirical software engineering* 22 (2017), pp. 1440–1477.

[17] Jean-Marc Jézéquel and Bertrand Meyer. "Design by Contract: The Lessons of Ariane." In: *IEEE Computer* 30 (Jan. 1997), pp. 129–130.

[18] Jozsef Katona. "Clean and dirty code comprehension by eye-tracking based evaluation using GP3 eye tracker." In: *Acta Polytechnica Hungarica* 18.1 (2021), pp. 79–99.

[19] Thomas D LaToza, David Garlan, James D Herbsleb, and Brad A Myers. "Program comprehension as fact finding." In: *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 2007, pp. 361–370.

[20] Chris Langhout and Maurício Aniche. "Atoms of Confusion in Java." In: *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. 2021, pp. 25–35.

[21] SeolHwa Lee, Andrew Matteson, Danial Hooshyar, SongHyun Kim, JaeBum Jung, GiChun Nam, and HeuiSeok Lim. "Comparing programming language comprehension between novice and expert programmers using eeg analysis." In: *2016 IEEE 16th international conference on bioinformatics and bioengineering (BIBE)*. IEEE. 2016, pp. 350–355.

[22] Stanley Letovsky and Elliot Soloway. "Delocalized plans and program comprehension." In: *IEEE software* 3.3 (1986), p. 41.

[23] Jung-Chun Liu, Kuei-An Li, Su-Ling Yeh, and Shao-Yi Chien. "Assessing perceptual load and cognitive load by fixation-related information of eye movements." In: *Sensors* 22.3 (2022), p. 1187.

[24] Thomas W MacFarland and Jan M Yates. "Kruskal–Wallis H-test for oneway analysis of variance (ANOVA) by ranks." In: *Introduction to nonparametric statistics for the biological sciences using R*. Springer, 2016, pp. 177–211.

[25] Flávio Medeiros, Gabriel Lima, Guilherme Amaral, Sven Apel, Christian Kästner, Márcio Ribeiro, and Rohit Gheyi. "An investigation of misunderstanding code patterns in C open-source software projects." In: *Empirical Software Engineering* 24 (2019), pp. 1693–1726.

[26] Roberto Minelli, Andrea Mocci, and Michele Lanza. "I know what you did last summer-an investigation of how developers spend their time." In: *2015 IEEE 23rd international conference on program comprehension*. IEEE. 2015, pp. 25–35.

[27] Patrick Peachock, Nicholas Iovino, and Bonita Sharif. "Investigating eye movements in natural language and c++ source code-a replication experiment." In: *Augmented Cognition. Neurocognition and Machine Learning: 11th International Conference, AC 2017, Held as Part of HCI International 2017, Vancouver, BC, Canada, July 9-14, 2017, Proceedings, Part I 11*. Springer. 2017, pp. 206–218.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[29] 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: *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2022, pp. 120–131.

[30] Norman Peitek, Janet Siegmund, and Sven Apel. "What drives the reading order of programmers? an eye tracking study." In: *Proceedings of the 28th international conference on program comprehension*. 2020, pp. 342–353.

[31] Norman Peitek, Janet Siegmund, Chris Parnin, Sven Apel, Johannes C Hofmeister, and André Brechmann. "Simultaneous measurement of program comprehension with fmri and eye tracking: A case study." In: *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*. 2018, pp. 1–10.

[32] Nancy Pennington. "Stimulus structures and mental representations in expert comprehension of computer programs." In: *Cognitive psychology* 19.3 (1987), pp. 295–341.

[33] Joseph Prusa, Taghi M Khoshgoftaar, and Naeem Seliya. "The effect of dataset size on training tweet sentiment classifiers." In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE. 2015, pp. 96–102.

[34] S. S. SHAPIRO and M. B. WILK. "An analysis of variance test for normality (complete samples)†." In: *Biometrika* 52.3-4 (Dec. 1965), pp. 591–611. ISSN: 0006-3444.

[35] Dario D Salvucci and Joseph H Goldberg. "Identifying fixations and saccades in eye-tracking protocols." In: *Proceedings of the 2000 symposium on Eye tracking research & applications*. 2000, pp. 71–78.

[36] Jeffrey D. Schall. "Frontal Eye Fields." In: *Encyclopedia of Neuroscience*. Ed. by Marc D. Binder, Nobutaka Hirokawa, and Uwe Windhorst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1635–1638. ISBN: 978-3-540-29678-2.

[37] Andrea Schankin, Annika Berger, Daniel V. Holt, Johannes C. Hofmeister, Till Riedel, and Michael Beigl. "Descriptive compound identifier names improve source code comprehension." In: *Proceedings of the 26th Conference on Program Comprehension*. ICPC '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, 31–40. ISBN: 9781450357142.

[38] Ben Shneiderman. "Measuring computer program quality and comprehension." In: *International Journal of Man-Machine Studies* 9.4 (1977), pp. 465–478.

[39] Mina Shojaeizadeh, Soussan Djamasbi, and Andrew Trapp. "Density of Gaze Points Within a Fixation and Information Processing Behaviour." In: vol. 9737. July 2016, pp. 465–471. ISBN: 978-3-319-40249-9.

[40] José Aldo Silva Da Costa and Rohit Gheyi. "Evaluating the Code Comprehension of Novices with Eye Tracking." In: *Proceedings of the XXII Brazilian Symposium on Software Quality*. 2023, pp. 332–341.

[41] Margarita Sordo and Qing Zeng. "On sample size and classification accuracy: A performance comparison." In: *International Symposium on Biological and Medical Data Analysis*. Springer. 2005, pp. 193–201.

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

[43] Marvin Wyrich. "Source Code Comprehension: A Contemporary Definition and Conceptual Model for Empirical Investigation." In: *arXiv preprint arXiv:2310.11301* (2023).

[44] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020.