Readability of Method Chains
A Controlled Experiment with Eye Tracking Approach

Pezhman Kasraee and Chong Lin

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden
This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:
Author(s):
Pezhman Kasraee
E-mail: me@pezhmankasraee.com
pezhmankasraee@gmail.com
Chong Lin
E-mail: chong.lynne@gmail.com
chong.lin@hotmail.com

University advisor:
Prof. Dr. Jürgen Börstler
Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Abstract

Context. Source codes with lower level of readability impose a higher cost to software maintainability. Research also exposed the importance of readability as a vital factor on software maintainability. Therefore, readability has recently investigated by software engineers. Readability involves human’s interactions making the study on readability difficult. In this study, we explore the readability of method chain and non-method chain in Java source codes with the means of an eye tracking device as a newly-introduced approach.

Objectives. Objectives of this study are: 1. we investigated if the number of methods in a method chain affects the readability of Java source codes, and 2. we investigated the readability of two programming styles: method chain and non-method chain.

Methods. To achieve both objectives of this study, two controlled experiments were conducted inside a laboratory with the means of an eye tracker device. In the first experiment, treatment groups were exposed separately to method chains with different number of methods. In the second experiment, the treatment groups were exposed separately to two different programming styles: method chain and non-method chain.

Results. Participants of this study were students with the average age of 24.56 years old. Fixation durations of participants’ reading was measured in millisecond (ms). In the first experiment, the average of fixation durations per method with lower number of methods was 600.93 ms, and with higher number of methods was 411.53 ms. In the second experiment, the average of fixation durations per method for non-method chain style was 357.94 ms, and for method chain style was 411.53 ms.

Conclusions. In the first experiment, the analysis of fixation durations indicates that method chains with higher number of methods are slightly more readable. Analysis of t-test (t-value = −0.5121, significance level = 0.05, and two-tailed probability) confirms that the results of the first experiment does not show a significant difference at p < 0.05. The results of the second experiment show that non-method chain style is slightly more readable in comparison with method chain style. Analysis of t-test (t-value = 3.1675, significance level = 0.05, and two-tailed probability) confirms that the results of the second experiment show a significant difference at p < 0.05.

Keywords: software engineering, eye tracking, readability, method chain.
Acknowledgement

Foremost, I would like to express my sincere gratitude to our supervisor Professor Dr. Jürgen Börstler who led us professionally, patiently and intelligently through this scientific journey. He is undoubtedly the origin of many ideas presented in this master’s thesis.

I would also like to thank my colleague and thesis partner, Chong Lin, for her true and frank collaboration, sharing knowledge and experience over the course of this study. Unquestionably, this study is the results of an extensive collaboration, mutual understanding and patience. Also, I would like to thank all great participants of this study who share their time voluntarily with us, and this study could not be accomplished without their valuable help.

My intellectual debt is to Professor Karl-Henrik Röbert, Professor Greg MacDonald, Dr. Ahmad Moeen-Taghavi, Dr. Alireza Zareekbatani, Dr. Mattias Eriksson and Jovin Hurry who have been great friends, courteous and diligent teachers and undoubtedly true motivations in my life and scientific journey.

I would like to thank my caring and lovely parents Masoud and Mahin who have supported me patiently and kindly during my whole life and encourage me strongly in my scientific endeavour; those who have given me a good notion of right values in life. Moreover, I would like to express my special thanks to my kind sister Pouya, and her brilliant husband Afshin and their son, Arman who has been a great resource of hope for me. Last but not least, I would like to express my deepest appreciation and love to Matilda who makes my life peaceful and glorious with her unconditional love.

Pezhman Kasraee
Karlskrona, Sweden.
9 May 2016.

Here I want to express my deep appreciation to our supervisor, Prof. Dr. Jürgen Börstler, for his encouragement, guidance and support for our work. I would also thank my friend and teammate, Pezhman, for his contribution and patience in our teamwork. I also thank the teachers of Department of Creative Technologies who helped us a lot in managing the lab and Tobii device. Without their support we could not complete the experiment.

I am thankful to the teachers and officers who offered us access and information on student resources, this helps us to improve our experiment.

Finally, I want to thank all the volunteers who participated our experiment, both in the pilot studies and the main experimentation, without your efforts we could not complete the study. Thank you.

Chong Lin
Karlskrona, Sweden.
9 May 2016.
## Contents

1 **Introduction** ........................................... 2  
   1.1 Motivations ........................................... 2  
   1.2 Key Concepts ........................................ 2  
      1.2.1 Identifiers ..................................... 2  
      1.2.2 Method Chains ................................ 4  
      1.2.3 Eye Tracking .................................... 7  
      1.2.4 Understanding vs. Understandability ........ 9  
      1.2.5 Understanding and Recognition .............. 10  
      1.2.6 Memorization .................................. 11  
      1.2.7 Maintainability ................................ 12  
      1.2.8 Texts: Natural Language Texts and Source Codes .... 13  
   1.3 Background .......................................... 14  
      1.3.1 Research on Readability in Literature ....... 14  
      1.3.2 Research on Readability in Software Engineering .... 17  
      1.3.3 Research on Readability in Software Engineering with Eye Tracking Approach ... 19  
   1.4 Contents ............................................. 22  
   1.5 Summary ............................................... 22  

2 **Related Work** ........................................ 23  
   2.1 Readability in Software Engineering ............ 23  
   2.2 Readability in Software Engineering with Eye Tracking Approach .................... 25  
   2.3 Summary ............................................... 27  

3 **Method** ............................................. 28  
   3.1 Aims and Objectives ............................... 28  
   3.2 Research Methodology .............................. 29  
   3.3 Readability ......................................... 31  
   3.4 Eye Tracking Laboratory ........................... 32  
   3.5 Selection of Participants ......................... 32
# List of Figures

1.1 The Key concepts of this study. ........................................... 3  
1.2 Tobii® T60. ................................................................. 7  
1.3 Saccades and fixations in a stimulus. ................................. 8  
1.4 Area of interest (AOI). .................................................... 8  
1.5 Tobii studio™ ........................................................... 9  
1.6 Physics formulas. .......................................................... 11  
1.7 Memorization process [70]. .............................................. 12  
1.8 Natural language text vs source codes. ............................... 13  
1.9 Contribution of other studies to this study. ......................... 16  

3.1 Contribution of the research questions to the aim of this study. 29  
3.2 Scope of the experiments A and B. ................................. 34  
3.3 Experiments A and experiment B and different variables. ...... 35  
3.4 Sample experiment materials for treatments MCT and MCS. 36  
3.5 MCT and MCS treatments in experiment A. ..................... 37  
3.6 Sample experiment materials for treatments MCS and Non-MCS. 38  
3.7 Treatments in both experiment A and experiment B. ............ 38  
3.8 Experiment path. ......................................................... 40  
3.9 Sitting posture. ............................................................ 42  
3.10 Contribution of treatments results to research questions. ...... 45  
3.11 Local Live Viewer setup. ............................................. 50  
3.12 Data validation process. .............................................. 53  

4.1 Histogram of total number of participants’ age. .................... 55  
4.2 Number of accepted and rejected participants in each step of recruitment and rejection. ................................. 56  
4.3 Histogram of eighteen accepted participants’ age. ............... 57  
4.4 Histogram of participants’ fixation durations in \( ms \) in the treatment MCT. ...................................................... 59  
4.5 Histogram of participants’ fixation durations in \( ms \) in the treatment MCS. .................................................. 61
4.6 Histogram of participants’ fixation durations in ms in the treatment Non-MCS. 

5.1 Comparison of participation sampling.

5.2 Comparison of participation sampling.

5.3 Comparison of sample mean of MCT and MCS.

5.4 Comparison of sample means of MCS and Non-MCS.

A.1 Warm-Up 1

A.2 Warm-Up 2

A.3 Warm-Up 3

A.4 Test 1, Method chain with two methods.

A.5 Test 2, Method chain with two methods.

A.6 Test 3, Method chain with two methods.

A.7 Test 4, Method chain with two methods.

A.8 Test 5, Method chain with two methods.

A.9 Test 1, Method chain with seven methods.

A.10 Test 2, Method chain with seven methods.

A.11 Test 3, Method chain with seven methods.

A.12 Test 4, Method chain with seven methods.

A.13 Test 5, Method chain with seven methods.

A.14 Test 1, Non-method chain with seven methods.

A.15 Test 2, Non-method chain with seven methods.

A.16 Test 3, Non-method chain with seven methods.

A.17 Test 4, Non-method chain with seven methods.

A.18 Test 5, Non-method chain with seven methods.

D.1 Participants outliers.

D.2 Test outliers of a particular participant.
### List of Tables

4.1 Number of accepted participants’ datasets in each treatment and test. ........................................ 57
4.2 Number of accepted participants’ tests in each treatment. ....................................................... 57
B.1 Table of Abbreviations. ................................................................. 100
Chapter 1

Introduction

1.1 Motivations

Code comprehension is a significant factor that can impose heavy price on software maintainability. This factor is about 50% of software maintenance costs [25, 46, 75], and maintenance costs between 40% and 70% of software life cycle [7, 28, 65]. Maintenance leads to modification of source codes [1]; thus readability should be regarded as an influential factor that affects the measurement of the maintainability [1]. Therefore, a source code with a higher degree of comprehensibility can reduce the costs of both software maintainability and software development life cycle; this, consequently, can lead to lower-cost software products. Code comprehension does not only affect the costs of maintenance, but the time which is also spent on maintenance. According to LoToza et al., reading source code takes more than 40% of comprehension time [38]. In 2015, Börstler et al. considered readability as an essential factor affecting understandability of source codes [11].

1.2 Key Concepts

In this part, we introduce the fundamental concepts and terminologies closely intertwined with the problem set of this study. A concise introduction regarding identifiers, method chains, eye tracking, understanding, understandability, memorization, maintainability and types of texts is absolutely indispensable to provide the readers with an insightful vision regarding this study. The key concepts of this study are presented in figure 1.1.

1.2.1 Identifiers

Identifiers are names selected by programmers to refer to a variable or method name, type, etc. Identifiers are significant factors affecting readability and understandability of source codes [4, 61]. Identifiers form nearly
70% of source codes [16, 17, 61]. Different programming languages may use different rules to create valid identifiers.

A common way to create a variable is to combine two or more chunks\(^1\) to create a descriptive name for an identifier. These compound names usually become long and difficult to read. Thus, software developers have introduced different naming conventions to make long identifiers more recognizable. Camel-case and underscore are two widely used approaches.

In **camel-case** naming convention, each chunk in identifiers starts with a capital letter expect the first letter that often starts with lowercase letters, e.g. `studentNameId`.

In underscore naming convention, an underline is used between each chunk to separate words, e.g. `student_name_id`.

```java
int dividendNumber = 10;
int divisorNumber = 5;
int quotativeDivision = calculateDivision(dividendNumber, divisorNumber);
```

Sample 1.1: A part of Java code that present division operation.

Sample 1.1 presents a simple division program in Java. The variable `dividendNumber` is composed of two chunks: `dividend` and `Number`. This variable denotes the numerator of a division.

The next variable, `divisorNumber`, is composed of two chunks: `divisor` and `Number`. This variable denotes denominator of a division, and the variable

\(^1\)A chunk can be a word in any languages, or an abbreviation.
quotativeDivision denotes the results of the division, and it is composed of two chunks: quotative and Division.

The calculateDivision method is composed of two chunks: calculate and Division, and it denotes the division operation.

1.2.2 Method Chains

Method chaining\(^2\) is a technique used in Object Oriented Programming (OOP) languages such as Java [12, 26]. In method chains, developers are unrestricted to only one method of an object in an expression\(^3\) [45]. That is to say, a developer can create an expression with multiple method invocations (method calls). This technique helps developers invoke a series of methods from a certain object, or as Börstler and Paech [12] stated method returning an object can be used to call another method [12]. As an example in Java, all the triggered methods of a certain object return the same object by using the return this; expression (see sample 1.5).

In method chains, the number of methods in a chain can increase to 6, 7 or more. By using method chains, the number of intermediate variables between different method invocations decreases [26]; moreover, method chain combines multiple method invocations only in one code expression [26].

Method chains are mainly used to create fluent interfaces\(^4\) [26]. Fluent interfaces use a domain-specific language (DSL) inside another programming language. DSL’s are specific programming languages aiming a particular domain of applications. SQL or CSS are two examples of domain-specific languages. DSL was coined in contrast to general-purposed languages (GPL’s) such as Java targeting a broader range of application domains. For instance, SQL eases the interaction of a Java-based application with a database. For example, Java is a host language embedding SQL commands inside Java codes to use a database.

We exemplify the difference between two programming styles: non-method chains (see sample 1.2) and method chains (see sample 1.3) with two simple source codes in Java.

We used intentionally the same method names to make the difference between two styles more understandable; however, these two source codes have different design and implementation in their respective classes.

---

\(^2\)Method chains are sometimes called as train wrecks, named parameter idiom, invocation chaining and chained invocation.

\(^3\)An expression ends in semi-colon ‘;’ in Java.

\(^4\)Fluent interfaces are sometimes called as internal or embedded DSL’s.
In both examples, first, we created an object (developer) from the Employee class, and then, we initialized the properties of the created class: developer. Non-method chains and method chains are used in sample 1.2 and sample 1.3, respectively.

```java
public static void main(String[] args) {
    /* Sample of non-method chain in Java. */
    Employee developer = new Employee();
    developer.setFirstName("John");
    developer.setSecondName("Doe");
    developer.setBirthYear(1976);
    developer.setEmployeeNumber("PK76SE");
    System.out.println(developer.toString());
}
```

Sample 1.2: Sample of non-method chain in Java.

In both source codes, first, the developer objects were created, but the difference is in initialization. In sample 1.2, the object, developer, is presented in every expression (in every line of code). However, in sample 1.3, the object and all initializer (setter) methods are presented only in one expression.

```java
public static void main(String[] args) {
    /* Sample of method chain in Java. */
    Employee developer = new Employee();
    System.out.println(developer.setFirstName("John")
        .setSecondName("Doe")
        .setBirthYear(1976)
        .setEmployeeNumber("PK76SE")
        .toString());
}
```

Sample 1.3: Sample of method chain in Java.

The difference between a non-method chain and method chain structure comes from different class design. While sample 1.4 represents a non-method chain class, the sample 1.5 presents a class that its methods can be used in a method chain style.

```java
public class Employee {

    private String firstName;
    private String secondName;
    private int birthDate;
    private String employeeNumber;

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```
public void setSecondName(String secondName) {
    this.secondName = secondName;
}

public void setBirthDate(int birthDate) {
    this.birthDate = birthDate;
}

public void setEmployeeNumber(String employeeNumber) {
    this.employeeNumber = employeeNumber;
}
}

Sample 1.4: A Java class that can create non-method chains in Java.

In sample 1.5, each method returns the current object by using the statement return this; Therefore, each return expression in a method is a pointer to the same object, so all elements of the object are again available through this method return. All methods modifies the properties of the same object.

For example, in the sample 1.3, the method setFirstName() returns the object developer by the statement return this; Since this method in sample 1.3 returns the whole object, the setSecondName() method can be attached to the previous method.

public class Employee {

    private String firstName;
    private String secondName;
    private int birthDate;
    private String employeeNumber;

    public Employee setFirstName(String firstName) {
        this.firstName = firstName;
        return this;
    }

    public Employee setSecondName(String secondName) {
        this.secondName = secondName;
        return this;
    }

    public Employee setBirthYear(int birthYear) {
        this.birthDate = birthYear;
        return this;
    }

    public Employee setEmployeeNumber(String employeeNumber) {
        this.employeeNumber = employeeNumber;
        return this;
    }
}
In sample 1.3, the number of expressions decreases to one expression by applying the method chain approach, and the whole expression is written only in one expression (see sample 1.3).

In comparison to the non-method chain approach, method chains can be written only in one line (methods can be next to each other, horizontally), and this technique can decrease the number of code lines, but it increases the length of the line. Locating the methods horizontally next to each other is not suggested, because Börstler et al. suggested that as the average sentence length increases the readability decreases [11].

1.2.3 Eye Tracking

Using an eye tracker is a recent approach to study the readability of source codes. In this method, the eye movements of participants are studied while they read the source codes.

![Figure 1.2: Tobii® T60.](image)

Two basic characteristic elements of eye movements are saccades and fixations.

Quick movements of eyes from one target to another target in a picture, book, etc. are called saccades [34, 49, 59, 60]. The period that eyes are kept steady in a certain location between two saccades is called fixation [34, 49, 59, 60]. In figure 1.3, green circles symbolize the fixations of the eye movements and the red lines between circles illustrate the saccades.
One of the variables, that Tobii® T60 measures, is the **fixation duration (FD)**. This variable is defined as the length of a gaze in millisecond (ms) at a specific location on a screen while a person reads a text.

![Figure 1.3: Saccades and fixations in a stimulus.](image)

Any images, texts and films used during experiments with an eye tracking device is called **stimuli**.

For a study, researchers usually divide the stimuli into different segments to study the eye movements of the participants on those particular areas. A particular segment is called **Area of Interest (AOI)**. That is, first, researchers identify some AOI’s on the stimuli, and then gather data regarding their participants’ eye movements from each identified AOI’s.

Figure 1.4 illustrates AOI in a sample stimulus. The captured data regarding the eye movements is captured from the red area in this stimulus.

![Figure 1.4: Area of interest (AOI).](image)

Studying of programmers’ eye movements helps software developers to produce source codes with higher level of readability [61].

---

**Chapter 1. Introduction**

---
In this study, we use Tobii® T60 Eye Tracker (see figure 1.2) to gather data from the participants’ eye movements.

**Gaze point** is another variable that Tobii® T60 provides. Gaze Point is the horizontal and vertical positions of the screen for either eye or the average for both eyes. This value is also employed for the fixation definition.

Tobii® device is equipped with a software called Tobii Studio™ (see figure 1.5) records data from each participant’s eye movements and stores the data in a respective participant’s file. This file is called a **dataset**. This software also helps us analyse the generated data.

![Tobii Studio™](image)

Figure 1.5: Tobii studio™.

### 1.2.4 Understanding vs. Understandability

**Understanding** and **understandability** are two different terms sometimes used incorrectly instead of each other. The similarities between both words can confuse novices. In this section, we explain each of these two terms to avoid any confusion in the rest of this study. At first, we define understandability, a term used in software engineering, and then, we define the word understanding exploited indirectly to define readability.

**Understandability**

Understandability is a term used commonly in software engineering. The term understandability was defined, narrowed down and well-established in software engineering canonical books, resources and standards such as *ISO/IEC TR 9126:2001* and *ISO/IEC 25010-2011*. 
According to ISO/IEC TR 9126:2001, understandability is a factor of usability that is a software quality [32]. After introduction of ISO/IEC 25010-2011, the new term appropriateness recognizability was used instead of understandability [33]. The newly-introduced term was considered as a more accurate term [33]. The term appropriateness recognizability is regarded as how users can recognize if a software product or system is suitable for their demands [33].

Software understandability is a feature of software quality [5, 36]. Software understandability means ease of understanding and to what extents software systems, components and their purpose are clear [5, 36, 66]. Boehm consider software understandability as a factor in software maintenance [5, 36, 66]. However, understanding is a general word with a broad meaning from a simple recognition to profound knowledge of a phenomenon.

Understanding

Understanding is a cognitive process that can be exemplified as a simple recognition of a shape or picture like a friend’s face or pattern, knowing the meaning of a word in a language, or how a machine works or how a natural phenomenon happens, or how a person thinks or behaves. It is sometimes difficult to distinguish between understanding and readability. Because these two concepts are very close, and they can affect each other [1, 11]. Our definition regarding readability and the difference between readability and understanding is presented in section 3.3.

Aggarwal et al. stated that understanding is a crucial factor to readability, and it is symbolized by comments in source codes [1]. Aggarwal et al. measured the readability of source codes by introducing the comment factor as the total line of codes per total line of comments [1]. In this study, we try to lessen the effect of understanding factor on readability. We discuss the readability in section 3.3.

1.2.5 Understanding and Recognition

In this study, we use the term recognition to define readability in section 3.3. The terms understanding and recognition are also two close terms that might confuse the readers. To avoid the readers’ confusion, we discuss the difference between understanding and recognition in this section.
We use *Einstein’s mass-energy equivalence* to ease the comprehension of this concept. People with different level of educations from a sixth-grade student to Albert Einstein have a different level of understanding regarding the equivalence $E = mc^2$. For example, a sixth-grade student can understand basic arithmetic operations and Latin letters, but Albert Einstein who discovered this equivalence had a very deep understanding regarding this concept.

Although different people have different level of understanding regarding this equivalence, all of these people can recognize this formula. For example, if the readers are provided with an image (see figure 1.6) that presents some physics formulas, they can recognize Einstein’s mass-energy equivalence among other physics formulas although they might have different level of understanding regarding this equivalence.

![Physics formulas](image)

**Figure 1.6: Physics formulas.**

We define recognition as the minimum time duration that a person requires to distinguish a shape, word, etc. among other elements of a set. We use this term to define the readability in section 3.3.

### 1.2.6 Memorization

All forms of intelligence work based on memory [71]. **Memorization** is one of the most important (dynamic) cognitive process of human being’s brain [70]. According to Wang [70], memorization includes two major steps: *establishment* and *construction* of the information in long-term memory (LTM).

He defined establishment in two steps as follows:

- **Encoding**: Presentation of the knowledge.
- **Retention**: Storing knowledge in LTM.

According to Wang [70], construction also consists of two parts as follows:

- **Retrieve**: Searching in LTM, and
• **Decoding**: Knowledge reformation.

Memorization is successful if the same information can be remembered correctly after a number of repetitions [70]. Memorization improves with repetition and practice [70]. The most important part of memorization process is to create or update LTM by searching and analyzing the contents of short-term memory (STM) and selecting the most frequently-used information into LTM [70].

Reitsma’s study on children between seven and eight years old in 1983 revealed that visual recognition of the unique graphemic composition of words plays a significant role to identify words in a text [57]. Word recognition becomes automatic by direct recognition of word building chunks [57, 76]. Rayner et al. stated that the mean fixation durations of words with higher frequency is shorter than infrequent words [55]. According to Jacobson and Dodwell and Rayner and Pollatsek, as texts become more complex in terms of typographical variables, fixation durations increase [54]. According to a study by Laufer in 1997, memorization, therefore, increases by higher word frequency because the subjects have more exposure to the words [39].

### 1.2.7 Maintainability

**Maintainability** is the most important quality factor (in software engineering) affected by readability [1, 25, 46, 75]. According to *ISO/IEC 25010-2011*, maintainability is defined as to what extent a product or system can be modified by maintainers. Moreover, this standard emphasizes that modification can occur in the form of corrections, improvements or adaptation of the software in requirements or functional specifications.

This process requires the software developers to go through the source codes to identify variables and method names to modify values, or improve or

---

5 The smallest meaningful units in a writing system as combination of letters.
change related algorithms. Developers usually use part of an identifier name to look for a certain identifier in the whole source code. They find all occurrence of the target variables or method names, and modify them based on the requests. Ease of reading or high readability can help developers to remember the identifier names and applications better. Aggarwal et al. stated maintenance in all sorts affects leads to source code modification [1]. Therefore, readability and understandability should be used as measures in maintainability [1].

1.2.8 Texts: Natural Language Texts and Source Codes

The concepts of text readability and understandability investigated by different researchers in different domains from linguistics to psychology and software engineering [19]. Readability is a measurement applied on written materials. In this study, we divided written materials into two following groups:

- **Natural language texts**: These texts are written with natural languages such as English, Swedish, etc. These languages have evolved throughout history of humankind. Novels, newspapers are mostly written in natural languages.

- **Source codes**: Software developers use programming languages such as C, Java, Python, etc. to create source codes. These written materials are usually in the form of digital files.

Both natural language texts and source codes share some common features although they have some differences. Natural language texts are used by human beings to interact with each other while the source codes are used to interact with (digital) machines.

![Figure 1.8: Natural language text vs source codes.](image)

*Syntax* and *semantics* are two common and substantial attributes in both groups of languages. In linguistics, syntax describes the correct order of
words\textsuperscript{6} in a sentence, and semantics concerns with the meaning of words and phrases. Although syntax and semantics are two common features in all languages, natural languages can use more flexible syntax and semantics in comparison to programming languages.

Prior to software engineers employ readability measurements on source codes, linguistics and literati measured the readability of natural language texts. Measuring readability has been applied in natural language texts long before the advent of software engineering concept, but this background contributes undoubtedly to providing a picture regarding readability of source codes in software engineering.

\section*{1.3 Background}

Readability is an interdisciplinary concept that has enthused researchers with diverse backgrounds from linguistics and psychologists to software engineers. In this section, we provide the readers with merely main research on readability aligned to the aim of this master’s thesis. Undoubtedly we were unable to cover all valuable research on the concept of readability, which are beyond the borders of the current study. We present the background of readability from three different perspectives: readability of natural language texts, readability of source codes and using eye tracking as an approach to study readability.

\subsection*{1.3.1 Research on Readability in Literature}

The early studies of English language readability started at the end of 19th century \cite{18}. Lucius Adelno Sherman is one of the pioneers who performed early research on the readability of English language texts \cite{19}. In 1893, he presented his statistical approach in the book \textit{Analytics of literature: A manual for the objective study of English prose and poetry} \cite{19, 62}. He discovered that the length of sentences in English prose in his time was shorter than the length of sentences in the Elizabethan \cite{62}.

Sherman investigated the change in the length of sentences by calculating the average number of words per sentence in English prose. He proposed that long sentences are more incomprehensible than short sentences \cite{62}. Readers tend to read longer sentences twice or more in order to perceive the meaning

\textsuperscript{6}In programming languages, words are in the forms of keywords (such as \texttt{for} and \texttt{while} in Java) or identifiers (such as variable and method names), etc.
of the sentences; this means, long sentences are less readable than short sentences.

Thorndike presented the frequency of English words for the first time in the book *The Teacher’s Word Book* [19]. This book contained 10,000 most frequently-used English words [64]. Each word was assigned with a credit number to reflect the word frequency. That is, higher credit number means the word is used more frequently.

*The Teacher’s Word Book* was regarded as a fundamental reference in early readability studies [19]. A large part of early studies [42, 68, 69] on English language readability was based on the Thorndike’s word list. That is to say, word frequency is regarded as an important factor that affects natural language readability.

Early studies on readability were mainly performed on primary and high school students [19, 42, 68, 69]. However, researchers explored the readability of reading materials for adults in 1930’s [19, 30, 47, 72]. For example, Gray and Leary investigated four groups of factors that affect the readability of texts for adults [18, 30]. These factors are summarized into four groups: 1. format or mechanical features, 2. general features of organization, 3. style of expression and presentation, and 4. content [30].

Lorge measured the readability of texts for children. Lorge introduced *Lorge Readability Index* in the paper *Predicting Readability* [19, 43]. However, this formula was also used to estimate the readability of texts for adults [19].

Two influential studies on readability including the Dale-Chall Readability Formula and Flesch Reading Ease Score (FRES) were published in 1948 [18]. According to Dubay [18], Dale-Chall formula and FRES are regarded as two reliable methods to measure the readability of a text; furthermore, FRES is one of the most-widely used and tested formula [18].

FRES is used to measure the readability of natural language texts [24, 18]. This method consists of two variables:

- **Average Sentence Length (ASL)**: The number of words per sentence.
- **Average Word Length (AWL)**: The number of syllables per words.

FRES quantifies the readability of an English text with a grade between zero and 100. A text with grade closer to 100 is easier to read than a text with grade closer to zero [24, 18].
Cloze test was introduced by Wilson Taylor to help measuring readers’ text comprehension [18, 63]. In a cloze test, several words were replaced with blanks in texts. Then, participants were asked to fill the blanks [18]. Cloze score is used to describe the percentage of correctly filled words [18]. Based on cloze scores, texts with higher scores are easier to read. The cloze test tool helps researchers to applied profound studies on readability [18].

Researches on readability revealed that readers’ idiosyncrasies (particular characteristics of an individual’s behaviour) would also affect the readability of texts. Some examples of idiosyncrasies are level of literacy, background knowledge of texts and interests of reading [18].

In the 1970s, the development of cognitive psychology and linguistics helped researchers discover more aspects of the reading process [18].

Epelboim et al. studied the role of whitespaces between words in texts [23]. In this study, they removed spaces utterly or substituted spaces with some other characters called fillers to investigate how this manipulation affects readability of texts [23]. The whitespaces were removed utterly or replaced with Greek or Latin letters, digits and shaded boxes\(^7\) [23].

\(^7\)Shaded box is a character defined in the Extended ASCII code equivalent to 176 in decimal or B0 in hexadecimal.
They found that removing or replacing spaces with fillers decreases the readability of texts [23]. However, replacing spaces between words with shaded boxes affects readability of texts less than the other fillers [23]. Rayner et al. also confirmed texts without spaces are difficult to read [56].

1.3.2 Research on Readability in Software Engineering

The concept of readability was attracted by software engineers in late 1960’s and in the beginning of 1970’s. Roberts [58] and McCracken et al. [44] are among the pioneers who studied the concept of readability in software engineering [35].

In 1976, James L. Elshoff investigated readability of 120 commercial PL/I software products, which were collected from General Motors’ computing installations. He claimed that most of the source codes have a low level of readability [20]. To improve the readability of programs and increase the efficiency of software development and maintenance, Elshoff suggested [20] the following instructions:

- Modularizing programs to avoid too big programs.
- Using more comments and better naming convention to increase readability.
- Exploiting a more controlled programming approach to decrease the complexity of programs to understand the logic and the flow of data in the programs better.
- Increasing programmers’ knowledge regarding the programming language that they use.

Later, Elshoff compared the source codes of two different types of PL/I programs written in two different programming paradigms\(^8\): one group of programs was developed with structured programming\(^9\) paradigm; while the other group of programs was developed with non-structured programming paradigm [21].

According to Elshoff [21], non-structured programming in comparison to structured programming has:

\(^8\)A method to categorize different programming styles. For example, structured programming, object oriented programming, functional programming, etc.

\(^9\)In structured programming, subroutines, block structures, for-while loops are used to avoid the use of goto expressions.
• less procedural statements,
• less \texttt{CALL} statements,
• less \texttt{ELSE} clauses,
• more \texttt{GOTO}, and
• more label statements.

He proposed that structured programming improves the readability of source codes and increases the quality of software products [21].

In 1982, Elshoff and Marcotty published a paper contained 13 suggestions that improve the readability of source codes written in PL/I programming language [22]. For example, they suggested that adding comments and avoiding \texttt{GOTO} statements increase the readability of PL/I programming language source codes [22]. Furthermore, they argued that the improvement of source code readability reduces the cost of software modification [22].

The importance of reading source code in software development is also agreed by other researchers like Deimel [15], Raymond [53] and Collar et al. [13].

Deimel stated that the ability of reading source codes is an important programmers’ skill that should be taught to all programming students [15]. He proposed that students should be taught how to read source codes in programming courses [15].

Deimel proposed three suggestions regarding how to teach code reading to students: first, the importance of source code reading should be emphasized; second, students should practice reading source code; third, guidelines on how to write readable codes should be taught to students [15].

Furthermore, Raymond stressed that reading source code is a substantial expertise in software maintenance [53], and also Collar et al. asserted that the improvement of software readability decreases the cost of software development [13].

Some researchers measured different characteristics of source codes such as identifiers, method chains and comments affecting the source code readability [4, 7, 11, 12, 51].

Buse and Weimer [7] proposed a model to predict the readability of source codes by using different characteristics of codes such as length of lines and identifiers, and also the number of keywords and identifiers, etc.
Binkley et al. investigated two different identifier naming styles and their effects on source code readability [4]. More elaboration regarding this study can be found in 2.1.

Posnett et al. argued that Buse and Weimer’s model was based on a limited size of source code snippets, and this model may not be able to predict the readability of large size source codes [51]. Posnett et al. improved Buse and Weimer’s model and proposed a simpler model to measure the readability of source codes [51].

Posnett’s model is based on the lines of code and token \(^{10}\)-level entropy [51]. Posnett et al. defined the entropy as the complexity or the degree of disorder. The token-level entropy describes relative distribution of tokens/characters in the source code [51].

Posnett et al. claimed the proposed model proved a better performance than Buse and Weimer’s model on source codes with larger sizes [51].

Börstler et al. proposed the Software Readability Ease Score (SRES) based on two variables: 1. average word length; 2. average sentence length to measure the readability of source codes [11]. More elaboration regarding this study can be found in 2.1.

Börstler and Paech questioned if the method chains and comments affect software readability and comprehension [12]. Over than 100 university students participated in this study [12]. They stated that they could not spot any statistical relations between the presence of method chains and readability of source code [12].

1.3.3 Research on Readability in Software Engineering with Eye Tracking Approach

Researchers started investigating the readers’ eye movements at the end of 1890’s and the beginning of 1900’s [31, 54]. They studied the readers’ eye movements during process of reading a text [31, 54].

In 1908, according to Huey [31], professor Louis Émile Javal, an ophthalmologist, was one of the pioneers who reported the basic characteristic elements of readers’ eye movement in reading [31].

\(^{10}\)Tokens are elements of programming languages such parenthesis, commas and loops etc.
In 1879, Javal stated the readers do not trace a line smoothly from beginning of the line to the end, instead eyes shift with some stops in between; this means eyes’ movements are a combination of short movements and stops [31]. When a person reads a text, the eyes move quickly from one location to another location in a short distance. The quick eyes’ movements are called saccades. Javal observed short stops between saccades when reading, these stops happen approximately with a frequency of ten characters [31]. These short stops are fixations.

Eye tracking technique was also applied in medical studies, psychology, etc. In software engineering, eye tracking techniques were applied in software usability, human-computer interaction, program comprehension, etc [3, 9, 10, 52, 67].

According to Sharafi et al. and Busjahn et al., Crosby and Stelovsky conducted the first study in software engineering with eye tracking technique in 1990 [9, 60]. In this study, they investigated how university students read algorithms written in Pascal [14]. They also compared the difference of reading patterns between second semester university students and graduated university students [14]. Crosby and Stelovsky concluded that reading source code is different from reading natural language texts [14]. They also found that programmers with high level of experiences concentrated more on areas where meaningful parts of codes are located [14].

Bednarik et al. introduced eye tracking technique as a valid way to study eyes’ movements of programmers. They investigated the reading behavior of high school and university students to comprehend source codes [3]. Their study showed that eye tracking techniques can help researchers to understand the program comprehension process of software developers [3].

Uwano et al. studied the eye movements of programmers to uncover the graduate students’ source code reading behavior with three to four years of programming experience [67]. They identified one reading pattern among participants called scan. That is to say, programmers read the whole source code in the beginning, and then they examine the source code deeper [67]. Uwano et al. measured the time that programmers spend to find errors in source codes. Therefore, they concluded that programmers spending adequate time on scanning might have higher performance in finding errors in source codes [67].

Sharif et al. repeated Binkley’s study with an eye tracker [61]. However, Sharif et al. did not report any difference between camel-case and underscore naming convention in understanding of source codes [61]. Furthermore,
they found that novices spent more time on searching a camel-case identifier than experts, while novices can find an underscore identifier faster than experts [61].

In 2011, Busjahn et al. explored the difference between reading natural language texts and reading source codes with an eye tracking approach [9]. Their results showed the variable global mean fixation durations (median of all FD’s) during reading natural language texts was shorter than reading source codes [9]. Readers were more likely to re-read texts while they read source codes in comparison to reading natural language texts [9].

In 2013, a group of researchers participated the first International Workshop on Eye Movements in Programming Education [52]. In this workshop, a group of researchers were invited to study the eye movements of two Java expert programmers. The eye movements of two experts were recorded by an eye tracker from SMI with the OGAMA tracking software while they read a Java source code with 23 lines [52]. Based on the results of the workshop, Busjahn et al. explored how eye tracking techniques can help improving the computer science education research [52].

The second International Workshop on Eye Movements in Programming Education [10] was held in 2014 in Germany. Similar to the first workshop, researchers were provided with two sets of eye movement recordings. In contrast to the first workshop, the researchers of this workshop studied the eye movements of two novice programmers. These two novice programmers attended a three-month Java course with six lessons at Freie Universität Berlin before this workshop [10]. Each set of eye movement recordings contains three recordings that was captured at the beginning, middle and end of the Java course [10].

Researchers observed that in the beginning of the programming course, novices read thoroughly the whole source code from the start to end frequently [10]. This reading pattern is similar to how people read natural language texts [10].

However, the source code reading behavior of novices changed drastically at the end of the programming course. First, novices read the whole source codes, then they jumped between lines to comprehend the codes in more details [10].

The use of eye tracking equipment helps software engineering researchers to gain a deeper view regarding the code reading process. Studying eye movements also help researchers to provide better educational materials regarding programming courses.
Sharafi et al. performed a systematic literature review regarding the studies that applied eye tracking as a tool in software engineering research [60]. Their study provided an overview of the current eye tracking studies, and also the measurements used to analyze eye tracking data [60]. Furthermore, the limitations of eye tracking technique and previous research were also presented in this study [60].

1.4 Contents

This thesis contains six chapters to refer to different perspective of this study. In chapter 1, introduction, the researchers’ motivation, necessary concepts regarding this research, history of readability is discussed. Chapter 2, related work, contains the most recent studies in software engineering or eye tracking forming the origins of this study. In chapter 3, method, aim and objectives, research questions and experiment design of this study is presented. Chapter 4, results, presents the results of the study without interpreting them. In chapter 5, analysis and discussion, the results of this study is interpreted, and finally, in chapter 6, conclusion, the importance of this study and future works is presented.

1.5 Summary

This section presented our motivation for the study as well as some key concepts that related with this study. The background part introduced the brief history of readability studies from literature to software engineering, and also we looked into some major studies on software readability with an eye tracking approach.
Chapter 2

Related Work

The researches elaborated in this section are the most relevant studies contributing directly to different parts of this study such as establishments of the research questions, the research and experiment design, and the conclusion. We present the studies in two groups: research with and without using eye tracking approach.

2.1 Readability in Software Engineering

In 2008, Buse and Weimer investigated the measurement of source code readability [7]. They were inspired by readability studies on natural languages texts [7]. For example, average word length and average sentence length are the metrics that are used in natural language texts [7]. Afterwards, they identified a set of metrics in source codes based on some already-available metrics in natural language texts [7]. Some examples of these easy-to-calculate metrics are length of lines, length of identifiers, number of keywords, number of identifiers, etc [7].

Buse and Weimer conducted a survey on readability of 100 short Java codes called snippets with average lines of 7.7 lines [7]. Buse and Weimer’s model applied machine learning based on 12,000 collected judgments from 120 computer science students on code readability.

They applied the following policies to select the snippets: the code snippets should be short, but with complete structure and logic [7]. By considering the mentioned policies, Buse and Weimer extracted code snippets based on a selection of three items from the following suggested statements: field declarations, assignments, function calls, breaks, continues, throws and returns that are the most basic units of Java programming language [7].

The code snippets should be simple to understand in order to minimize the effects of judgments from code comprehension aspects [7]. The participants should be able to understand easily the provided algorithms or functions in
the snippets. They designed a model to define readability of source codes based on humans’ judgments [7].

By considering Buse and Weimer’s experiment, we intend to lessen the complexity of source codes to only investigate the readability of method chains in Java.

Binkley et al. investigated if identifier naming styles affect source code readability [4]. In programming, identifiers are names for variables and methods. Identifiers are elaborated in section 1.2.1.

Binkley et al. studied more than six million lines of code from open-source applications, and they extracted approximately 50 million identifiers out of the source codes [4]. Identifiers are important elements in programming languages affecting readability and understandability of source codes [4, 61].

Binkley et al. studied the importance of identifier naming conventions on the readability of source codes [4]. Based on studies from natural language texts and psychology, underscore naming convention is more readable than camel-case style [23, 56].

However, according to Binkley et al., participants achieved more accurate results in searching specific identifier names with camel-case style than underscore naming convention [4]. As well as, Java naming convention standard provided by Oracle, Binkley et al.’s study helped us select the naming convention for the identifiers of our experiment design.

Börstler et al. argued that measurements such as FRES [24] are likely to be inapplicable to measure software readability, because source codes have almost different features making them distinguishable from natural language texts [11]. Different methods such as SRES [11] and PHD [51] are offered to quantify the readability of software source codes.

In the SRES method, Börstler et al. claimed that AWL should be longer than ASL in source codes [11]. That is to say, longer identifiers improve source codes comprehensibility [11, 41]. However, they suggested that shorter ASL can enhance readability [11, 40]. Furthermore, they also argued that ASL is a more significant factor in comparison with AWL [11]. Method chaining is one of the approaches that can produce long expressions.

In 2016, Börstler and Paech investigated how method chains and comments affect software readability and comprehension [12]. They conducted a study with 104 university students [12]. According to Börstler and Paech [12], software readability and comprehension can be measured by:
• **Perceived readability**: Participants measured the readability of the source codes with five difficulty levels from very easy to very difficult according to their personal experience.

• **Reading time**: The time spent on reading the source codes in seconds.

• **Performance**: To measure participants’ comprehension, percentage of correct answers were calculated after reading the source codes.

Börstler and Paech could not observe certain relations between the existence of method chains and the source code readability [12]. Börstler et al.’s studies stimulated our interests in investigating how method chains affect the readability of source codes; this led us to question if a method chain with more methods decrease the readability of source code. We also wanted to investigate if non-method chains are more readable than method chains.

### 2.2 Readability in Software Engineering with Eye Tracking Approach

Eye tracking technique was introduced as a new approach to investigate how programmers comprehend source codes in recent years. Many studies explored software comprehension and readability through eye tracking approach [3, 9, 10, 52, 61, 67].

Bednarik et al. investigated the source code reading of participants with the means of eye tracking method [3]. Their study showed that eye tracking is a valid approach to investigate the behavior of source code reading [3]. They also provided the methodology to analyse eye tracking data. This methodology provided us with some ideas to analyse the data from an eye tracker in our experiment.

Sharif et al. investigated which naming conventions: camel-case or underscore is more readable [61]. This topic was also investigated by Binkley et al. previously [4]. Sharif et al. repeated Binkley et al.’s experiment with an eye tracking approach [61]. They stated that eye tracking helps researchers to gain additional proof on naming convention related studies [61]. Sharif et al. found no difference between camel-case and underscore with respect to source code comprehension, but programmers distinguished underscore identifiers quicker than the camel-case identifiers [61].

Sharif et al.’s study also supported the selection of camel-case naming convention in our experiment design.
Busjahn et al. stressed that reading source codes is a different behavior from reading natural language texts [8]. They stated that texts with lower level of readability cause longer FD’s in comparison to texts with higher level of readability [8]. Therefore, we regarded FD as an indicator to measure the readability of source codes in our experiment. They compared reading behavior of novices and experts on source codes and natural language texts [8]. They observed that reading source codes does not follow the same pattern similar to reading natural language texts [8].

They found that natural language texts with Latin alphabets (such as texts in English) are read from left to right and from top to bottom. Readers read natural language texts almost line-by-line which means with high linearity. This reading order is called *story reading*.

They also noticed that novices almost follow the same reading order when they read source codes, but with less linearity. That is to say, novices sometimes did not read the source codes line-by-line. The reading order that novices read source codes is called *story order* [8].

When expert programmers read a source code, they follow highly the program sequence order (with less linearity) although they still follow natural language text reading order; the reading style that is used by expert programmers is called *execution order* [8].

Linearity describes how similar the order of source code readers’ eye movements is to story order [8]. The following metrics were used to measure the linearity:

- **Local metrics** includes gaze point, regression\(^1\), saccades etc.

- **Global metrics** is the order of fixations, which reveals the readers follow a story order or execution order when reading.

Busjahn et al. proposed that this study can be applied in both industry and education, because the results of this study contribute to developing more comprehensible codes [8].

Sharafi et al. conducted a systematic literature review on 37 eye tracking studies in software engineering [60]. They extracted and summarized the number of participants in each study. Their results showed that the mode number of participants in these studies was 15 and the median was 18 [60]. They categorized the studies into five groups based on the topic of studies [60]:

\(^1\)The backward saccades in reading.
• model comprehension,
• debugging,
• code comprehension,
• collaborative interactions, and
• traceability.

Their study presented the current metrics and limitations in eye tracking techniques. The metrics used to measure eye tracking data were classified into four categories [60]:

• metrics based on the number of fixations,
• metrics based on fixation durations,
• metrics based on saccades, and
• metrics based on scan-paths$^2$.

One of the common metrics is the average fixation durations (AFD) that is also used by Busjahn et al. [9] and Sharif et al. [61]. We use fixation durations to measure the readability of method chains in Java.

Sharafi et al. reported 1. inconsistency in terminology, metrics and methods names in the current studies using eye tracking in software engineering, and also 2. replication of studies are hardly possible because most of the studies did not share their experiment design and data [60]. At the end, they provided suggestions for researchers who are beginners in software engineering research with eye tracking techniques. Sharafi et al.’s study provides us with an integral picture of the current studies regarding eye tracking in software engineering.

### 2.3 Summary

This section introduced the most relevant researches contributing to our study. The Related Work section was formed with two sub-sections: readability studies in software engineering with and without eye tracking approach.

This study investigates the readability of method chains in Java with the means of eye tracking. This research opens another horizon on readability of method chains. Börstler and Paech [12] conducted a study on readability of method chains without using eye tracking technique.

$^2$Chronological order of fixations.
Chapter 3

Method

3.1 Aims and Objectives

Investigating a study from different perspectives is an approach to build a more solid foundation for science. Börstler and Paech [12] conducted a study on readability of method chains. Studies conducted by Bednarik et al., Busjahn et al., and Sharif et al. stated that eye tracking can be used as an instrument to study the behaviour of programmers’ source codes reading and comprehension [3, 8, 61].

Therefore, another study with a close concept to Börstler and Paech’s study but with a different approach such as eye tracking can contribute to a deeper understanding regarding this problem.

The aim of the present study is to investigate the effects of method chains on readability of source codes by the means of eye tracking. We use an eye tracking device to observe the participants’ eye movements to investigate readability of method chains.

To achieve this aim, we defined the objectives of this study as follows: 1. we investigate the existence of relations between the number of methods in a method chain and readability of Java source codes, and 2. we investigate the readability of two programming styles: method chain and non-method chain.

Generated data from an eye tracking device provides a better picture that describes if any relations exist between the number of methods in a method chain and the fixation durations captured from participants’ eye movements.

We formulated the following two research questions (RQ’s) to achieve the proposed aim and objectives of this study:

- **RQ 1**: Which relations exist between the number of methods in a method chain and readability?
• **RQ 2**: In which way do method chains and non-method chains affect readability of source codes with respect to readers’ eye movement?

To answer RQ 1, we conduct one experiment to answer if any relations exist between number of methods in a Java method chain and readability of source codes by using an eye tracking device. We calculate AFD per method in method chains with different number of methods, and then by comparison of the results, we investigate to what extent increase or decrease in the number of methods in a method chain may affect the readability of method chains in Java.

![Figure 3.1: Contribution of the research questions to the aim of this study.](image)

RQ 2 questions if any difference exists between method chains and non-method chains in respect to readability. We compare two programming styles: method chains and non-method chains by the means of an eye tracker. We investigate if AFD’s per method differ while participants read the two programming styles. Finally, we compare AFD’s per method for both styles, and study to what extent different programming styles may affect the readability of methods in Java.

### 3.2 Research Methodology

In this section, we discuss our motivation for the selection of research methodology. Before starting a study, it is important that researchers know restrictions and possibilities of all research methods [74]. Then, the researchers can opt for the best possible research method that is adaptable to the study circumstances. Research in software engineering can be performed by applying 1. scientific, 2. engineering, 3. empirical, and 4. analytical methods [2, 27, 74].

In scientific approach, scientists observe the world and propose a model or theory. Then, they simulate the model, measure and analyze it, and finally they should validate the hypotheses [27, 74].

In engineering approach, which is common in industry, scientists study the existing solutions. They build or develop, measure, analyze and evaluate these current solutions [27, 74].
Empirical approaches are used where scientists are unable to address the problem with laws of nature; this approach is broadly used in social sciences [74]. Scientists propose a model, develop a statistical method, and validate it [27, 74].

Finally, in analytical approach, scientists propose a formal theory and if possible, they compare it with other empirical observations [27, 74].

Software engineering and social sciences share one common feature: both concern with human behaviour. Human beings are critical resources in software development; therefore, human behaviour is a significant concern that is addressed by researchers in software engineering [74]. Therefore, software engineering researchers use widely empirical studies similar to their counterparts in social sciences. In this study, we apply an empirical study because we investigate human behaviour: the eye movements of our subjects.

Two different (research) paradigms are available to address the empirical studies [74]:

- **Exploratory**: This research performs in natural contexts of the phenomenon, and data are gathered through observations; this research demands qualitative research\(^1\) [74].

- **Explanatory**: The aim of the research is to find a causality relationship between two or some groups, and also it is used to quantify a relationship. This research demands quantitative research\(^2\) [74].

In this study, we plan to describe the relations between two variables and perform comparisons between groups of participants. Thus, we use a quantitative research paradigm to answer our research questions.

Four different strategies are available to researchers to perform an empirical study: 1. survey, 2. case study, 3. (controlled) experiment and 4. quasi-experiment [74].

Survey is widely applied when researchers study the users’ experience of a specific tool or technique [74]. In a survey, researchers are unable to control measurements, execution of the study or manipulation of variables. In survey, results can be compared with the similar studies [74]. In this strategy, researchers collect qualitative\(^3\) or quantitative\(^4\) data by using questionnaires or conducting interviews [74].

---
\(^1\)Flexible research design.
\(^2\)Fixed design.
\(^3\)Data that are measured with their qualities e.g. smells, tastes, colours, etc.
\(^4\)Data that are measured in numbers, e.g. number of students in a classroom.
In software engineering, a case study is a strategy explaining a software engineering phenomena by observation in the natural context of the phenomena [48, 74].

Controlled experiments are applied to investigate a relation: a variable is manipulated while the other variables or factors are kept constant during the experiment, and the outcome variable is measured [74]. Participants in controlled experiments are assigned to random treatments\(^5\) or conditions [74].

The difference between controlled experiment and quasi-experiment is in assignment of treatments to participants [74]. In quasi-experiment treatments are not assigned to subjects of the study randomly [74].

In this study, we have one independent variable that is manipulated during the experiment – number of methods in a method chain – and the dependent variable which should be measured – fixation duration (FD).

To be able to perform this study, we should find a relation, in a controlled environment, and the subjects of the study should assigned to random treatments. Therefore, to answer our RQ’s, we conduct a controlled experiment.

### 3.3 Readability

Based on the provided explanation in the understanding, memorization and maintainability sections, we consider readability in this study as the time duration that a person requires to recognize a word or a pseudo-word or a non-word.

Here, recognition means the minimum time duration that a reader requires to distinguish correctly a word or a pseudo-word or a non-word.

The term correctly is used, because if a word is recognized incorrectly, it means that the reading process has not accomplished successfully. Therefore, the result of reading of a word should always be compared with a reliable lexicon, and the result of reading of a pseudo-word should always be compared with a previously provided answer key to assure that the reader has read the words or texts correctly.

Minimum time duration is the minimum duration of time that a person requires to scan a word without thinking about or trying to understand the word or pseudo-word meaning.

\(^5\)A group of conditions that is designed for a specific experiment.
For example, the word *barn*\textsuperscript{6} in Swedish and English can be read correctly although it has different meanings in both languages. Apart from how speakers of these two languages perceive or understand the word barn, both English and Swedish language speakers can read the word. A pseudo-word like *pemi-raxico* can be read although it does not have any meaning at least in English language.

### 3.4 Eye Tracking Laboratory

The environment of controlled experiments is usually inside a laboratory, and controlled experiments are employed when the researchers need control over the situation [74]. The following factors were considered during the experiment:

- Eye tracking device setting.
- Laboratory light setting.
- Quite place without interruption.
- Alerting signs to avoid possible interruptions.

### 3.5 Selection of Participants

Recruitment of participants is a significant factor that affects the generalization of the study [74]. The selected participants should represent the population that participants are selected from [74]. To conduct this study, we use *convenience sampling* to recruit the participants of this study [74]. This technique belongs to *non-probability sampling techniques* [74]. In this technique, the most convenient and nearest people are selected to participate in the study [74].

The participants of this study are volunteer students from Blekinge Tekniska Högskola\textsuperscript{7}. These participants must only and only participate in one of the available treatments of this study. Sharafi et al. examined 37 eye tracking studies in software engineering, and summarized the number of participants for each study [60]. Their results showed the mode number of participants in these studies was 15 and the median was 18 [60].

\textsuperscript{6}Barn means a child or children in Swedish

\textsuperscript{7}Blekinge Institute of Technology, Karlskrona, Sweden.
3.5.1 Participants’ Assignment Strategy

We assign the participants to each treatment (in both experiments) based on the following factors to keep the equality among different groups. To be able to maximize randomization over the participants. Before each test, we assess:

- if they wear any forms of glasses or contact lenses,
- their mother tongues, and
- genders.

3.6 Selection of Variables and Measurements

According to Wohlin et al., two groups of variables should be considered in an experiment [74]. These groups are summarized as follows:

- **Independent variables (Factors):**
  
  - **Controlled factors:** the factors that are controlled and have fixed values during the experiment.
  
  - **Manipulatable factors:** the ones that their values are changed during the experimentation, and they are assigned with manipulatable values. The value of a manipulatable variable is called a treatment.

- **Dependent variable:** The effect of change in independent variables on dependable variables should be studied. In this study, only one dependent variable exists.

**Experiments in This Study**

We performed two similar experiments in this study. In this study, we call these two experiments: **experiment A** and **experiment B** (see 3.2).

These two experiments are similar, because we measure the same dependent variable: FD (see figure 3.3).

However, these two experiments are different because they have different manipulatable factors (see figure 3.3). In experiment A, the manipulatable factor is *number of methods*, while the manipulatable factor is *programming style* in experiment B.

Experiment A and experiment B provide the results for RQ 1 and RQ 2, respectively.
List of Controlled Factors

Both experiment A and B share the same independent variables with fixed values. These variables are enlisted as follows:

- **Number of chunks in method identifiers**: According to Binkley et al., around 90% of identifiers include one, two or three chunks [4]. We keep the number of chunks constant and equal to three chunks. During the pilot study, we noticed that if we keep the number of chunks less than three, the result might be indistinguishable and, with the method identifiers more than three chunks, results would be less generalizable.

- **Length of methods**: We used source codes from the open source community to be able to generalize our study further. Therefore, we examined nine different computer games written in Java, and we extracted 216 method names. The average length of extracted method identifiers are 15.60 characters. Thus, we kept the length of the methods constant and equal to 15 characters to be able to close to the existing source codes in the open source community.

- **Name of methods**: Name of the methods were used without any change from the downloaded computer games from the open source community.

- **Length and name of the objects**: We kept the name of the object name constant in all tests and treatments to avoid any extra influence on the study. We used the common name of obj for the object that has the length of three characters.

- **Source code appearance**: All the tests used the font Consolas, and the font size of 12. To create codes, first, we used Sublime Text 2, version 2.0.2 on Microsoft Windows 7 platform, to create Java source codes. Then, we created images from the Java files. We used these images in the eye tracker. The created images have the size of 1280 × 1024 pixels.
All source code pictures are aligned vertically and horizontally in the center of each image.

- **Style of identifiers**: We use and select the method identifiers that have the camel-case style. Sharif et al.’s study suggested the underscore naming style is recognized faster than the camel-case naming style [61]; this helps us to achieve more distinguishable data while we study fixation durations of the participants by the eye tracker.

- **Code style**: Based on the method identifiers that we extracted from nine computer games from open source community, we observed that method chains are often presented in a vertical format. Therefore, both method chains and non-method chains are presented in a vertical format. Sample of the experiment materials are presented in section 3.8.2, and all the experiment materials are presented in appendix A.

![Figure 3.3: Experiments A and experiment B and different variables.](image)

**Dependent Variable**

The only dependent variable of this study is FD belonging to an *interval scale*. This variable is measured in millisecond. Busjahn et al. regarded the variable FD as an indicator of readability [8, 9]. Fixation is defined as a relative\(^8\) stillness of eyes during a certain period of time when the participants stare at a specific location on the screen [34, 49, 59, 60]. Fixation duration is a period of time that fixation lasts (see section 1.2.3).

Eye tracking studies in psychology revealed that cognitive efforts mostly happened during fixation durations rather than saccades [60]. A longer fixation duration means the subject has more cognitive efforts during that time [3].

\(^8\)Eyes move slightly even if this movements are unrecognizable or insensible by eye trackers.
Difficult texts usually indicates longer fixation durations [9, 54, 61]. By considering the reasons above, we decided to use fixation duration as an indicator to measure the readability of method chains in Java.

Fixation durations can be calculated by several methods, one of the most common method is **average fixation duration (AFD)** [9, 60, 61].

Therefore, we use the sample mean of FD’s per method as a metric to measure the readability of method chains in Java. More details regarding the calculation of average fixation durations in this study is presented in section 4.3.

**Treatments in Experiment A**

We conducted experiment A to answer RQ 1. Participants of this experiment are divided into two groups.

![Sample experiment materials for treatments MCT and MCS.](image)

In this experiment, the manipulatable factor is the number of methods. Therefore, the treatments are designed based on different values of the manipulatable factor of this experiment. These two treatments are called **treatment MCT**\(^9\) and **treatment MCS**\(^10\) (see figure 3.5). One group is assigned to treatment MCT, and another group is assigned to treatment MCS. The results of these two treatments are calculated and compared.

In this study, number of methods in a method chain is limited to only two or seven, because 1. the length of method chains hardly exceed more than six or seven methods in real source codes. So, they represent the highest and lowest number of methods in method chains; 2. we avoid using method chains with

---

\(^9\)Abbreviation for Method Chains with Two Methods  
\(^10\)Abbreviation for Method Chains with Seven Methods
length between two and seven, because the pilot study showed that providing distinguishable data with close number of methods would be very difficult or impossible. That is to say, to make the results more distinguishable, we use the minimum and maximum possible number of methods in method chains.

Therefore, the first group of participants are exposed to tests including method chains with two methods (MCT). A sample code from treatment MCT is presented in sample 3.1. Another group of participants are exposed to tests including method chains with seven methods (MCS). A sample code from treatment MCS is presented in sample 3.2.

All source codes regarding the treatments MCT and MCS are presented in appendix A.

**Treatments in Experiment B**

This experiment provides the answer to RQ 2. Participants of experiment B are divided into two groups.

In experiment B, the manipulatable factor is the programming style. Therefore, the treatments are designed based on different values of the manipulatable factor of experiment B. The first group should be exposed to tests with method chains, and the second group should be exposed to tests with non-method chains called treatment Non-MCS\(^{11}\). Finally, the results of these two treatments are calculated and compared.

However, we intentionally exposed our participants to only one treatment, because as we described, we perform two similar experiments, and these two experiments has similar dependent variable that is FD (see figure 3.7).

\(^{11}\)Abbreviation for Non-Method Chains with Seven Methods
Figure 3.6: Sample experiment materials for treatments MCS and Non-MCS.

Figure 3.7: Treatments in both experiment A and experiment B.
Thus, the necessary data regarding the method chains from treatment MCS in experiment A is available, and consequently, it is used to be compared with the results of the treatment Non-MCS from experiment B. Therefore, we perform experiment B with only one group, and the result of experiment B (with only treatment Non-MCS) is compared with the result from treatment MCS in experiment A. A sample code from treatment Non-MCS is presented in sample 3.3.

In this study, we have only one variable to measure: FD (see section 1.2.3). This variable is measured in millisecond (ms).

All source codes regarding the treatments MCS and Non-MCS are presented in appendix A.

3.7 Experiment Operation

3.7.1 Experiment Protocol

This experiment is divided into three main steps:

- introduction to the experiment,
- warm-up and
- main experiments (experiment A and experiment B).

Introduction to the experiment and warm-up is two fixed parts of the experiment operation. Although the main experiments have different tests and slightly different treatments, the process of experiment execution is fixed.

In the beginning of the experimentation session, lack of information regarding the experimentation might put the participants in a stressful, uncomfortable or uncertain situation. Introduction to the experiment was designed to provide the participants with basic necessary information. This helped the participants to overcome stress originated largely from their lack of knowledge concerning the experiment.

The aim of the warm-up is to ensure that participants know the basics that they need to conduct the experiment [37].

In this part of the experiment, we must ensure that our participants know:

- the necessary terminology that might be used during the experiments,
- the environment of laboratory,
the eye tracking system and

- the tasks.

In main experimentation, our participants performed their predefined tasks while we measure the variables that we planned to capture during experiments. To decrease the risks and likelihood of errors during the main experiments; a guideline is provided. To read about this guideline, see section 3.7.2. The participants should examine the provided Java source codes inside the stimuli. Each participant looks through the provided Java source codes stimulus by stimulus.

### 3.7.2 Experiment Guideline

The instructions in the guideline are provided based on 1. this study researchers’ observation during different pilot studies, 2. learning from the similar studies, and 3. the manuals provided by the Tobii company before the final experiments.
Introduction to the Experiment

In the beginning of the experimentation session, lack of information regarding the experimentation might put the participants in a stressful, uncomfortable or uncertain situation. To overcome this problem, we start providing the participants with:

- a very short introduction regarding the aim of this study,
- what eye tracking device is, and how it works,
- terminology used during the experimentation (e.g. calibration), and
- tasks that they should perform during the experimentation session.

After giving some general information regarding the experiment, we provide the participants with necessary and practical information regarding their sitting posture in front of the eye tracking device (Tobii® T60 is used in this study, see figure 1.2). Obviously, this information is critically essential to this study. A wrong sitting posture leads to wrong eye calibration; this influences negatively the results of the experiments. The most important instructions are listed as follows:

- **Distance from the eye tracking device**: Participants should keep the eyes between 58 and 62 cm from the eye tracking device (see figure 3.9).

- **Sitting position**: Participants should sit in an upright position (see figure 3.9).

- **State of eyes**: Participants should keep the eyes wide and open while staring at the display during the experimentation. Participants should not look anywhere out of the display frame.

- **No head or body movements**: Participants should avoid any unnecessary head or body movements during the experiments.

- **Process of calibration**: Participants should follow a red light on the screen with their eyes as they keep the sitting position constantly during the calibration or recordings.

- **Think-aloud technique**: Participants should announce the answers to the proposed questions for each test aloud, as the researchers (observers) do not notice when the participants have found the right answer.
Warm-Up 1: Exploring a Picture

The aim of warm-up experiments is to familiarize the participants with the experimentation process. When participants understand what an eye tracking device is, their curiosity might increase. They might be eager to know how this device works and what kind of services it can provide. This situation might increase their excitement and interests regarding the eye tracking device.

Therefore, in the first warm-up experiment, we show them a picture and ask them to find some certain elements in that picture. With this experiment, we aim to overcome two challenges: 1. to decrease participants’ excitement raising from their curiosity, and 2. to practice the sitting posture, and calibration process.

- **Aim:** To decrease the participants’ curiosity, practice the sitting posture and increase their comfort.

- **Method:** A very simple picture is presented to the participants, and they are asked to count the number of a certain item in the picture (see figure 3.8).

- **Steps:**
  - **Question:** How many cats are in the next picture?
  - **Picture:** See figure A.1
  - **Answer:** Participants should say the answer aloud. The correct answer is *three.*
• **Timing:** We give them enough time to satisfy their curiosity: maximum of three minutes.

• **Consideration 1:** Errors during this part are unimportant, and we ignore them.

• **Consideration 2:** We repeat the guidelines if we notice the participants’ mistakes.

• **Consideration 3:** They might forget to announce that they have found the answer.

**Warm-Up 2: Reading an English Language Text**

The aim of this experiment is to familiarize the participants with the process of the main experiments. At this stage, participants are asked to read an English language text and announce occurrence of a specific word (see figure 3.8).

The difference between the warm-up 2 and the main experiments is the applied language. In this step, we use English language, while in the main experiments, Java programming language is used.

The participants should read a simple English language text, and they should immediately announce loudly the line number corresponded to the location of the key word.

• **Aim:** To make the participants familiar with the process of the main experiments. Participants should understand that saying the line number of key word loudly is of paramount importance to this study.

• **Method:** A very simple English language text is presented to the participants, and they should find the key word, and announce the relevant line number aloud as soon as they find it in the text (see figure 3.8).

• **Steps:**
  - **Sample:** An example that provides the participants with short information regarding warm-up 2.
  - **Question:** In which line number is the word fish located?
  - **Picture:** See figure A.2.
  - **Answer:** Participants should say the answer aloud. The correct answer is line number 16.
• **Timing:** We give them enough time to satisfy their curiosity: **maximum of three minutes.**

• **Consideration 1:** Errors during this part are unimportant, and we ignore them.

• **Consideration 2:** We repeat the guidelines if we notice the participants’ mistakes.

• **Consideration 3:** They might forget to announce that they have found the answer.

**Warm-Up 3: Method Chain**

The aim of this warm-up is to provide the participants with some basic information regarding what a method chain is.

In this warm-up experiment, participants practice the process in the main experiment. Participants should look for a specific method identifier and say the line number aloud.

• **Aim:** To familiarize the participants with the main experiments (see figure 3.8).

• **Method:** Participants see the method identifier as a key to look for. Afterwards, they find the specific method identifier in the method chain (see figure A.3).

• **Steps:**
  - **Sample:** An example that provides the participants with short information regarding method chain (see figure A.3).
  - **Question:** Find the method `ifStateProvide()`.
  - **Source Code:** See figure A.3.
  - **Answer:** Participants should say the answer aloud. The correct answer is *line number three*.

• **Timing:** We give them enough time to find the location of the answer in the provided source code on the display **maximum of three minutes.**

• **Consideration 1:** Errors during this part are unimportant, and we ignore them. We should ask them to pay more attention to the guidelines.
• **Consideration 2:** We repeat the guidelines if we notice the participants’ mistakes.

• **Consideration 3:** They might forget to announce that they have found the answer.

**Main Experimentations**

*Generalization* of results and *consistency* of tests are two factors that we consider the most to create our tests.

To answer RQ 1, we designed two different treatments:

• *Treatment MCT:* This treatment includes five tests with method chains with two methods.

• *Treatment MCS:* This treatment includes five tests with method chains with seven methods.

![Figure 3.10: Contribution of treatments results to research questions.](image)

To answer RQ 2, we use the results of the treatment MCS, and we also conduct treatments: Non-MCS; this treatment includes five tests with non-method chains with seven methods. In the main experiment, each treatment is exposed to the participants with five tests or trials instead of only one test; this provides us with more reliable confidence interval and results in more reliable data. We attempted to a large extent to keep the conditions of the experiments and experiment environment quite fixed and without any change during all five tests. We divided our participants into three groups. The first group was exposed to treatment MCT, the second group was exposed to treatment MCS, and the third group was exposed to treatment Non-MCS.

General instructions before any treatments in the main experiment are as follows:
• **Timing:** We give them enough time to find the answer: **maximum of three minutes**.

• **Consideration 1:** Errors during this part are critically important. Before performing the main experiments, participants are asked to pay more attention to the guidelines.

• **Consideration 2:** We do not repeat the guidelines during the main experiments.

• **Consideration 3:** They might forget to announce that they have found the answer.

During the main experiments the following risks are considered:

• **Head and body movements:**
  
  – **Solution 1:** During the introduction and warm-up sessions, we repeat the sitting posture guideline.
  
  – **Solution 2:** After the main experiment, we check the variable *validity codes* generated by Tobii® device for both eyes. This variable helps us assess if the data that we captured by the eye tracker for a participant is valid. If more than 95% of the values of this variable for all five tests are valid, we accept the result of the experiment. Here, valid means at least one of the eyes should contain a valid value (see section 3.9 for more information).

• **Forget to announce line number:**
  
  – **Solution 1:** During the introduction and warm-up sessions, we repeat guideline.
  
  – **Solution 2:** If this behaviour repeats more than two times during the main experiments, the result of the experiment would be rejected.

• **Wrong answers:**
  
  – **Solution 1:** During the introduction and warm-up sessions, we repeat the guideline.
  
  – **Solution 2:** If this behaviour repeats more than two times, the result of the experiment would be rejected.

• **Forget to answer as soon as they found the answer:**
Chapter 3. Method

- **Solution 1:** During the introduction and warm-up sessions, we repeat the guideline.

- **Solution 2:** It is difficult to catch this problem. However, if we observe that the participant insist intentionally or unintentionally on the mistake during the warm-up or the introduction session, we cancel the whole experiment. It is possible to observe this mistake during the warm-up because we are supposed to have a conversation with our participants.

**Treatment MCT**

The first group of our participants are exposed to this treatment. Treatment MCT is composed of five tests. The aim of this treatment is to measure the fixation durations of participants while they read method chains with two methods. The tests are presented to participants in a randomly-selected order. In this treatment, participants are asked to find a method identifier, and then, they should look for the specified method identifier in the source code.

- **Aim:** To provide part of the answer to RQ 1.

- **Method:** To find a method identifier in a method chain.

- **Steps:**
  - Test 1:
    * **Question and Source code:** See figure A.4.
    * **Answer:** The answer is *method number two*.
  - Test 2:
    * **Question and Source code:** See figure A.5.
    * **Answer:** The answer is *method number one*.
  - Test 3:
    * **Question and Source code:** See figure A.6.
    * **Answer:** The answer is *method number two*.
  - Test 4:
    * **Question and Source code:** See figure A.7.
    * **Answer:** The answer is *method number one*.
  - Test 5:
* Question and Source code: See figure A.8.
* Answer: The answer is method number two.

- **Timing:** We give them enough time to read each sample code: maximum of three minutes.

- **Consideration 1:** Errors during this part are critically important.

- **Consideration 2:** We cannot distract participants’ attention by giving any sort of instruction.

**Treatment MCS**

The second group of our participants are exposed to this treatment. Treatment MCS is composed of five tests. The aim of this treatment is to measure the fixation durations of participants while they read method chains with seven methods. The tests are presented to participants in a randomly-selected order. In this treatment, participants are asked to find a method identifier, and then, they should look for the specified method identifier in the source code.

- **Aim:** To provide part of the answer to RQ 1 and RQ 2.

- **Method:** To find a method identifier in a method chain.

- **Steps:**
  - Test 1:
    * Question and Source code: See figure A.9.
    * Answer: The answer is method number four.
  - Test 2:
    * Question and Source code: See figure A.10.
    * Answer: The answer is method number seven.
  - Test 3:
    * Question and Source code: See figure A.11.
    * Answer: The answer is method number two.
  - Test 4:
    * Question and Source code: See figure A.12.
    * Answer: The answer is method number four.
  - Test 5:
• **Timing:** We give them enough time to read each sample code: maximum of three minutes.

• **Consideration 1:** Errors during this part are critically important.

• **Consideration 2:** We cannot distract participants’ attention by giving any sort of instruction.

**Treatment Non-MCS**

The third group of our participants are exposed to this treatment. Treatment Non-MCS is composed of five tests. The aim of this treatment is to measure the fixation durations of participants while they read non-method chains with seven methods. The tests are presented to participants in a randomly-selected order. In this treatment, participants are asked to find a method identifier, and then, they should look for the specified method identifier in the source code.

• **Aim:** To provide part of the answer to RQ 2.

• **Method:** To find a method identifier in a sample with non-method chain style.

• **Steps:**
  
  – Test 1:
    
    * **Question and Source code:** See figure A.14.
    
    * **Answer:** The answer is *method number four*.
  
  – Test 2:
    
    * **Question and Source code:** See figure A.15.
    
    * **Answer:** The answer is *method number seven*.
  
  – Test 3:
    
    * **Question and Source code:** See figure A.16.
    
    * **Answer:** The answer is *method number two*.
  
  – Test 4:
    
    * **Question and Source code:** See figure A.17.
    
    * **Answer:** The answer is *method number four*. 
– Test 5:
  * Question and Source code: See figure A.18.
  * Answer: The answer is method number five.

- **Timing:** We give them enough time to read each sample code: maximum of three minutes.

- **Consideration 1:** Errors during this part are critically important.

- **Consideration 2:** We cannot distract participants’ attention by giving any sort of instruction.

### 3.8 Instrumentation

#### 3.8.1 Measurement Instrument: Eye Tracking Device

We use Tobii® T60 Eye Tracker to observe the participants’ eye movements in this study.

According to Tobii device manual, data rate\(^{12}\) for Tobii® T60 Eye Tracker is 60 Hz. We use *Local Live Viewer* setup for Tobii® T60 Eye Tracker (see figure 3.11); this setting is used when the researchers supervise the experiment live from another display next to the eye tracker.

![Local Live Viewer setup](image)

**Figure 3.11:** Local Live Viewer setup.

This setting is usually used when the behaviour of the participant should be observed or with think-aloud approach – a method that participants look for the answer in the eye-tracking display and answer the proposed questions verbally. This setting includes two displays, one with embedded eye trackers that are connected to a computer via USB port, and the second display that connect to the computer via VGA port (see figure 3.11).

During the experiment, participants sit in front of the eye tracker device, and the researchers use the second display. The participants perform the tasks after the necessary calibrations.

\(^{12}\)Number of recorded gaze points per second.
3.8.2 Experiment Materials Design: Code Samples

We design three treatments (MCT, MCS and Non-MCS; see 3.6) for both experiments of this study. Each treatment contains five tests including sample codes with method chains or non-method chains in Java codes. General rules that we applied to design the sample codes (tests) are based on the following constrains:

- Buse and Weimer proposed a model to measure the readability of source codes [7]. They designed a set of principles for selecting source codes in their experiment. We designed the guidelines for our code samples in this study, by considering the source code selection policy in Buse and Weimer’s study.

- We minimize the complexity of source codes to only focus on investigating the readability of method chains in Java. Since the context of source codes is a factor that have important influence on readability, we should minimize its possible effect on our study. We intend to investigate the readability of method chains instead of understandability or comprehension. Therefore, we should minimize the comprehension efforts of participants during the experiments.

- The source codes are designed purely with method chains or non-method chains expressions without statements, comments, body of methods, etc. to decrease the complexity of source codes and only focus on readability of method chains.

- We defined the controlled factors of these two experiments (see 3.6), based on the following factors:
  
  - *Number of chunks in method identifiers* is fixed, and equal to three.
  - *Length of methods* is fixed with 15 characters (17 characters with a pair of parentheses).
  - *Length and name of the objects* is fixed as obj.
  - *Source code appearance* is fixed.
  - *Style of identifiers* is fixed as camel-case style.
  - *Code style* is fixed as vertical format for all the treatments.

- To create the tests, we examined nine different computer games written in Java from open source community (regarding the length of method identifiers see 3.6).
Chapter 3. Method

- We used a random-generator to put the method identifiers in a random order; this also includes the answers.

Detailed information for the controlled factors is presented in section 3.6 and also the background information about this part is presented in the related work: chapter 2.

**Code Samples for MCT**

A sample code (test number 1) from treatment MCT is presented as follow:

```java
1  obj.getUpgradesFrom()
2       .oldestWaitStart();
```

Sample 3.1: Sample of a method chain with two methods for treatment MCT.

By observing the source codes from open source community, we used a vertical format to design the experiment materials (see section 3.6). Therefore, each method is presented in one separated line of code in MCT.

**Code Samples for MCS**

A sample code (test number 2) from treatment MCS is presented as follow:

```java
1  obj.getExportAmount()
2       .getTilePosition()
3       .glTexParameteri()
4       .addModelMessage()
5       .getUnknownEnemy()
6       .createRootSheet()
7       .nexueServerHost();
```

Sample 3.2: Sample of a method chain with seven methods for treatment MCS.

By observing the source codes from open source community, we used a vertical format to design the experiment materials (see section 3.6). Therefore, each method is presented in one separated line of code in MCS.

**Code Samples for Non-MCS**

A sample code (test number 3) from treatment Non-MCS is presented as follow:
1. `obj = obj.getUnitIterator();
2. obj = obj.getImageLibrary();
3. obj = obj.testDoubleScore();
4. obj = obj.getMapGenerator();
5. obj = obj.getSeasonNumber();
6. obj = obj.getFeatureIndex();
7. obj = obj.getSelectedTile();

Sample 3.3: Sample of a method chain with seven methods for treatment Non-MCS.

By observing the source codes from open source community, we used a vertical format to design the experiment materials (see section 3.6). Therefore, the object and the respective method is presented in one separate line in Non-MCS.

Identifiers in sample codes in both MCS and Non-MCS are the same, and presented in the same order, but they have different programming styles. Methods are represented with the method chain style in MCS, while the methods follow the non-method chain style in Non-MCS.

### 3.9 Data Validation

After conducting the experimentation process, we should validate the data. The captured data for each participant is stored in one dataset. Each dataset contains 60 records for every second of recording process. For example, if the average of recording for a particular participant lasts 60 seconds per test, this specific participant’s dataset contains approximately 18,000 records of data.

![Figure 3.12: Data validation process.](image)

To validate the captured data, we designed the following two steps:

- **Rejection of datasets**: Datasets are rejected based on the percentage of errors for all records in a dataset.

- **Rejection of tests**: Tests are rejected if the gazing is outside of the predefined AOI’s. It might consequently lead to rejection of a dataset if all five tests are rejected.
3.9.1 Rejection of Datasets

Rejection of a dataset, simply means that all five tests of a particular participant are rejected because of one or two following reasons:

- if a participant cannot provide the correct answers to any of five questions in a treatment; or

- if the provided results from the eye tracker device (the variable validity codes) for a specific participant showed that more than 5% of the recordings are invalid.

3.9.2 Rejection of Tests

After rejection of datasets, we examine the accepted datasets. Each dataset contains FD’s for both inside and outside of AOI’s.

Each dataset provides a variable called AOI[AOIName]Hit for each gaze point. This binary variable\(^{13}\) can be assigned only to zero or one. \(\text{AOI}[\text{AOIName}]\text{Hit} = 1\) means that the respective gaze point occurred inside the AOI, and consequently, \(\text{AOI}[\text{AOIName}]\text{Hit} = 0\) means that the respective gaze point occurred outside the AOI.

In each dataset, all values of \(\text{AOI}[\text{AOIName}]\text{Hit} = 0\) were removed. That is to say, we remove all gazing points that are outside of the defined AOI’s.

After this process, some of the tests were rejected because all values of \(\text{AOI}[\text{AOIName}]\text{Hit}\) related to one particular test were equal to zero.

Briefly, a test is rejected if:

- a participant answers the question wrongly; or

- all values of \(\text{AOI}[\text{AOIName}]\text{Hit}\) related to one particular test are equal to zero.

If all values of the variable \(\text{AOI}[\text{AOIName}]\text{Hit}\) for all five tests were equal to zero, the specific dataset would be rejected, because it means that none of the gaze points occurred inside the defined AOI’s.

\(^{13}\text{A variable with only two values.}\)
4.1 Participants

The participants of this study were 31 volunteer students from Blekinge Tekniska Högskola selected based on non-probability sampling techniques. Each volunteer participated only and only in one of the available treatments. The participants of this study were novices with no industry-level programming background. Approximately 30% of the participants were females.

Figure 4.1 illustrates the histogram of the participants’ age with respect to measures of central tendency. This diagram also provides the sample mean ($\bar{x}$) and median of the participants’ age as the measures of central tendency (see figure 4.1). Section 3.5 provides more details regarding the recruitment of participants.

![Histogram of participants' age](chart.png)

**Figure 4.1: Histogram of total number of participants’ age.**

The total number of participants had the sample arithmetic mean of $\bar{x} = 25.4839$, and median of 24 years old. Since

$$\bar{x} > \text{median}$$
this diagram (figure 4.1), therefore, is slightly skewed to the right\footnote{When a diagram skewed right, it means that the mean is greater than median.}, because of the existence of some outliers among the age of the participants.

## 4.2 Data Validation

Before the validation process, 31 participants attended the experiments, and consequently, the eye tracker generated a dataset for each corresponding participant.

![Data Validation Diagram](image)

**Figure 4.2:** Number of accepted and rejected participants in each step of recruitment and rejection.

In the step one of validation (rejection of datasets), 12 datasets (participants) were rejected because of invalid data. The rejection of datasets were elaborated in section 3.9. This means that 19 datasets out of 31 were accepted at the end of this step.

The step two of the validation process (rejection of tests) was started with 19 accepted datasets from the step one of the validation process.

The rejection of tests was elaborated in section 3.9.2. Total of nine tests out of all datasets were rejected. Five tests out of nine rejected tests belonged to one particular dataset (participant); consequently, we rejected one more dataset. The total number of accepted datasets (participants) was 18 at the end of the second step of the validation process.

The number of accepted tests in each treatment grouped by gender and treatment is presented in table 4.1.

Diagram 4.3 depicts the age of the participants that their results were accepted.
Chapter 4. Results

Figure 4.3: Histogram of eighteen accepted participants’ age.

<table>
<thead>
<tr>
<th></th>
<th>MCT</th>
<th>MCS</th>
<th>Non-MCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Female Participants</td>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Male Participants</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Total of Participants</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 4.1: Number of accepted participants’ datasets in each treatment and test.

After the validation steps, finally, 18 datasets were accepted for analysis. The number of accepted tests in each treatment is presented in the table 4.2.

The sample mean of the accepted participants’ age was $\bar{x}_{\text{accepted}} = 24.5556$ and the median was 24 years old.

<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3</th>
<th>Test 4</th>
<th>Test 5</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCT</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>23</td>
</tr>
<tr>
<td>MCS</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>29</td>
</tr>
<tr>
<td>Non-MCS</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>34</td>
</tr>
</tbody>
</table>

Table 4.2: Number of accepted participants’ tests in each treatment.

In the rest of this chapter, all analysis are presented only based on the 18 accepted datasets.

4.3 Method to Calculate the Average of Fixation Durations per Method for Each Treatment

In this section, we present the way that we calculated the sample mean of FD’s per method for a particular treatment. The sample mean of FD’s per method for the treatment MCT is denoted by $\bar{x}_{m_{\text{MCT}}}$, for MCS is denoted by
\( \bar{x}_{m_{\text{MCS}}} \), and for Non-MCS is denoted by \( \bar{x}_{m_{\text{NonMCS}}} \). To calculate the sample mean of FD’s per method for each particular treatment, we use the following steps:

Each element of this formula is defined as follows:

- First, we calculated the summation of all FD’s for all (accepted) tests\(^2\) in a particular treatment, e.g. MCT; this is denoted as \( \sum FD \) in the formula.

- The number of methods in a treatment is denoted as \( n(M) \). For example, all the method chains in the treatment Non-MCS are composed of seven methods, and consequently, for Non-MCS, the number of methods is seven, which is presented as \( n(M) = 7 \).

- The summation of total number accepted tests in a particular treatment, and the result of this test is valid. This denoted with \( \sum_{i=1}^{5} n(T_i) \).

In this formula, \( i \) presents the test number. Maximum number of tests in a particular treatment is five.

\( T \) presents a Test. Therefore, \( T_2 \) presents the test number 2 from a particular treatment. For example, according to table 4.2, the number of accepted Test 2 in Non-MCS is six, and it is denoted as \( n(T_2) = 6 \). Consequently, we must calculate the summation of all numbers of accepted tests in one particular treatment. Therefore, \( \sum_{i=1}^{5} n(T_i) \) for Non-MCS (see table 4.1) is calculated as follows:

\[
\sum_{i=1}^{5} n(T_i) = n(T_1) + n(T_2) + n(T_3) + n(T_4) + n(T_5) = 7 + 6 + 7 + 7 + 7 + 7
\]

The summation of all numbers of accepted tests in Non-MCS is denoted by \( \sum_{i=1}^{5} n(T_i) = 34 \).

Mathematical presentation of the sample mean of FD’s per method for a particular treatment is denoted as:

\[
\bar{x}_m = \frac{\sum FD}{n(M) \sum_{i=1}^{5} n(T_i)}
\]

\(^2\)In accepted tests, gazing has occurred inside the AOI
4.4 Result of Experiment A

The result of the experiment A is presented in this section. Experiment A was designed to provide the answer to RQ 1. This experiment is composed of two treatments: MCT and MCS\textsuperscript{3}.

4.4.1 Result of MCT Used to Answer RQ 1

In this part, we present the sample mean of FD’s per method in MCT. It provides part of the answer to RQ 1. The sample mean of FD’s per method in MCT was calculated based on the presented method in section 4.3.

The sample mean of FD’s per method in MCT is $\bar{x}_{m,MCT} = 600.9348 \text{ ms}$. That is to say, each participant in MCT scanned a method in 600.9348 ms on average. We also calculated the average number of fixations based on Sharafi et al.’s suggestion [60]. The average number of fixations per AOI for MCT is 3.9565.

4.4.2 Result of MCT Used for Validation

Histogram of FD’s for MCT is illustrated in figure 4.4. Furthermore, the sample mean ($\bar{x}_{h,MCT}$), median and sample standard deviation ($s_{h,MCT}$) of the histogram of participants’ fixation durations are also presented. The presented results in this section are used to validate the recording data during MCT. Total number of five participants including four males and one female participants conducted the tests in MCT.

![Histogram of participants’ fixation durations in ms in the treatment MCT.](image)

\textsuperscript{3}This treatment is shared between two experiments.
Analysis of the histogram (see figure 4.4) presents that $\bar{x}_{h_{\text{MCT}}} = 303.7692 \text{ ms}$, with the median of 250 ms, and $s_{h_{\text{MCT}}} = 188.4870$.

$$\bar{x}_{h_{\text{MCT}}} > \text{median}$$

The comparison of $\bar{x}_{h_{\text{MCT}}}$ and the median of the histogram of participants’ fixation durations shows that the histogram of participants’ fixation durations of MCT is skewed to the right.

### 4.4.3 Result of MCS Used to Answer RQ 1 and RQ 2

In this part, we present the sample mean of FD’s per method in MCS denoted by ($\bar{x}_{m_{\text{MCS}}}$). It provides part of the answer to RQ 1 and RQ 2. The sample mean of FD’s per method in MCS was calculated base on the presented method in section 4.3.

The sample mean of FD’s per method in MCS is $\bar{x}_{m_{\text{MCS}}} = 411.5320 \text{ ms}$. That is to say, each participant in MCS scanned a method in 411.5320 ms in average. We also calculated the average number of fixations based on Sharafi et al.’s suggestion [60]. The average number of fixations per AOI for MCS is 9.1379.

### 4.4.4 Result of MCS Used for Validation

Histogram of FD’s for MCS is illustrated in figure 4.5. Furthermore, the sample mean ($\bar{x}_{h_{\text{MCS}}}$), median and sample standard deviation ($s_{h_{\text{MCS}}}$) of the histogram of participants’ fixation durations are also presented. The presented results in this section is used to validate the recording data during MCS. Total number of six participants including five male and one female participants conducted the tests in MCS.

Analysis of the histogram (see figure 4.5) presents that $\bar{x}_{h_{\text{MCS}}} = 315.2491 \text{ ms}$, with the median of 266 ms, and $s_{h_{\text{MCS}}} = 183.1216$.

$$\bar{x}_{h_{\text{MCS}}} > \text{median}$$

The comparison of $\bar{x}_{h_{\text{MCS}}}$ and the median of the histogram of participants’ fixation durations shows that the histogram of participants’ fixation durations of MCS is skewed to the right.
4.5 Result for Experiment B

The experiment B designed to answer RQ 2. In this experiment, two treatments are considered: MCS⁴ and Non-MCS. The result of the treatment MCS was discussed in section 4.4.4.

4.5.1 Result of Non-MCS Used to Answer RQ 2

In this part, we present the sample mean of FD’s per method in Non-MCS denoted by \( \bar{x}_{m_{NonMCS}} \). It provides part of the answer to RQ 2. The sample mean of FD’s per method in Non-MCS was calculated base on the presented method in section 4.3.

The sample mean of FD’s per method in Non-MCS is \( \bar{x}_{m_{NonMCS}} = 357.9412 \text{ ms} \). That is to say, each participant in Non-MCS scanned a method in 357.9412 ms in average. We also calculated the average number of fixations based on Sharafi et al.’s suggestion [60]. The average number of fixations per AOI for Non-MCS is 9.1471.

4.5.2 Result of Non-MCS Used for Validation

Histogram of fixation durations for Non-MCS is illustrated in figure 4.6. Furthermore, the sample mean (\( \bar{x}_{h_{NonMCS}} \)), median and sample standard deviation (\( s_{h_{NonMCS}} \)) of the histogram of participants’ fixation durations are also presented. The presented result in this section is used to validate the recording data during Non-MCS. Total number of seven participants including five male and two female participants conducted the tests in Non-MCS.

---

⁴This treatment is shared between two experiments.
Figure 4.6: Histogram of participants’ fixation durations in ms in the treatment Non-MCS.

Analysis of the histogram (see figure 4.6) presents that $\overline{x}_{h_{\text{NonMCS}}} = 273.9228$ ms, with median of 250 ms, and $s_{h_{\text{NonMCS}}} = 128.6077$.

$$\overline{x}_{h_{\text{NonMCS}}} > \text{median}$$

The comparison of $\overline{x}_{h_{\text{NonMCS}}}$ and the median of the histogram of participants’ fixation durations shows that the histogram of participants’ fixation durations of Non-MCS is skewed to the right.
Chapter 5

Analysis and Discussion

5.1 Participants

After the validation process (see section 4.1), the histogram of participants’ age with accepted datasets became more symmetric. If we compare the sample standard deviations of all participants’ age ($s_{all} = 4.2651$) and accepted participants’ age ($s_{accepted} = 2.9945$), we notice that

$$s_{all} > s_{accepted}$$

The comparison above also confirms that the age of participants with the accepted datasets are closer to a normal distribution. (see figure 5.1).

![Figure 5.1: Comparison of participation sampling.](image)

The diagram 5.2 confirms that the participants’ age includes less extreme outliers after the rejection process. The interquartile range (IQR) (see figure 5.2) is a bigger area after the rejection of the participants. This makes the participants’ age distribution closer to normal distribution.
5.2 Experiment A

To answer the first research question (RQ 1), we use the result from the experiment A including two treatments of MCT and MCS.

According to the result presented in section 4.4, we identified that the sample mean of FD’s per method for the treatment MCT is $\bar{x}_{m_{MCT}} = 600.9348 \text{ ms}$ and the sample mean of FD’s per method for the treatment MCS is $\bar{x}_{m_{MCS}} = 411.5320 \text{ ms}$.

Comparison of the sample mean of both treatments MCT and MCS shows that $\bar{x}_{m_{MCT}} > \bar{x}_{m_{MCS}}$ (see figure 5.3).

Analysis of AFD’s shows that each method in MCS takes shorter time to be scanned in comparison to each method in MCT.
The covariance of number of methods and FD’s was -236.7535. The covariance shows the number of methods and FD’s have a reverse relation because $\text{Cov}(n(M_{mc}), \bar{x}_m) < 0$, where $n(M_{mc})$ denotes the number of methods in a method chain, and $\bar{x}_m$ denotes the sample mean of FD’s per method.

To assess the result of this experiment, we used the sample standard deviations of both treatments. Histograms of both MCT and MCS are presented in section 4.4.2 and section 4.4.4, respectively.

In MCT histogram (see figure 4.4), the sample standard deviation and sample mean are represented as follows: $s_{hMCT} = 188.4870$ and $\bar{x}_{hMCT} = 303.7692$ ms.

In MCS histogram (see figure 4.5), the sample standard deviation and sample mean are as follows: $s_{hMCS} = 183.1216$ and $\bar{x}_{hMCS} = 315.2491$ ms.

To calculate the error function, we use the following formula:

$$s_{\text{max}} = \max(s_{hMCT}, s_{hMCS})$$

$$z = \frac{|\bar{x}_{hMCT} - \bar{x}_{hMCS}|}{2\sqrt{s}}$$

We calculated z-score for experiment A as $z_A = 0.4181$. Then, the error is calculated with the following function:

$$\text{erf}(z) = (\sqrt{\pi}) \int_0^z \exp(-y^2) \, dy$$

$\text{erf}(z_A) = 0.4456$ indicates that the result of this study with probability of 44% confirms that the analysis of this study is not based on chances.

AFD in method chains with two and seven methods does not show a considerable difference in millisecond.

$$\Delta_{x_A} = |\bar{x}_{mMCT} - \bar{x}_{mMCS}| = 189.4028 \text{ ms}$$

Furthermore, the complementary error function shows that result of this experiment with a probability of 55% might be based on coincidence (chance).

$$\text{erfc}(z_A) = 1 - \text{erf}(z_A) = 1 - 0.4456 = 0.5544 \approx 55\%$$

The covariance confirms that the existence of a negative relation between number of methods in method chains and fixation durations. To answer RQ 1, we conclude that as the number of methods increases, fixation durations decrease. Therefore, the readability of method chains increases.
By considering $z_A = 0.4181$, the result of t-test analysis with

\[ \text{significance level} = 0.05 \]

and two-tailed probability of $\pm z$ yields $t-value = -0.5121$, and the t-test analysis shows that the difference is not significant at $p < 0.05$.

People might have different speed of reading. Fast and slow readers might affect the results of experiment A. To study the effect of fast and slow readers on the results of experiment A, we performed an extra analysis to identify the fast and slow readers of this experiment. For more information regarding the process of slow and fast readers’ identification, see appendix D.

By removing the outliers, AFD per method for treatment MCT is $586.6786 \text{ms}$, and AFD per method for treatment MCS is $416.2773 \text{ms}$. The results of this study after removing the outliers (fast and slow readers) still show that $\bar{x}_{mMCT} > \bar{x}_{mMCS}$, and it confirms that outliers do not affect the conclusion from experiment A.

5.3 Experiment B

To answer the second research question (RQ 2), we use the results from the experiment B including two treatments of MCS and Non-MCS. According to the result presented in section 4.5, we identified that the sample mean of FD’s per method for the treatment MCS is $\bar{x}_{mMCS} = 411.5320 \text{ms}$ and the sample mean of FD’s per method for the treatment Non-MCS is $\bar{x}_{mNonMCS} = 357.9412 \text{ms}$.

![Figure 5.4: Comparison of sample means of MCS and Non-MCS.](image-url)
Chapter 5. Analysis and Discussion

By considering the sample mean of both treatments, we notice that

$$\bar{x}_{MCS} > \bar{x}_{NonMCS}$$

Therefore, $$\bar{x}_{MCS}$$, $$\bar{x}_{NonMCS}$$ and also the diagram 5.4 confirm that the participants of this study scan the non-method chains slightly faster than method chains; this means that the readability of the non-method chains might be higher than method chains.

To assess the result of this experiment, we used the sample standard deviations of both treatments. MCS and Non-MCS are presented in section 4.4.4 and section 4.5.2, respectively.

In MCS histogram (see figure 4.5), the sample standard deviation and sample mean are represented as follows: $$s_{hMCS} = 183.1216$$ and $$\bar{x}_{hMCS} = 315.2491$$ ms.

In Non-MCS histogram (see figure 4.6), the sample standard deviation and sample mean are as follows: $$s_{hNonMCS} = 128.6077$$ and $$\bar{x}_{hNonMCS} = 273.9228$$ ms.

We calculated z-score for experiment B as $$z_B = 1.5270$$. The value of error function is $$\text{erf}(z_B) = 0.9692$$; this means that the results of this study with probability of 97% confirms the relations between different programming styles of method chains and non-method chains.

Analysis of AFD’s in method chains and non-method chains does not show a considerable difference in millisecond.

$$\Delta \bar{x}_B = |\bar{x}_{MCS} - \bar{x}_{NonMCS}| = 53.5908$$ ms

However, the complementary error function shows that the result of this experiment with only probability of 3% is based on chance.

$$\text{erfc}(z_B) = 1 - \text{erf}(z_B) = 1 - 0.9692 = 0.0308 \approx 3\%$$

Therefore, by considering the result of the complementary error function, the result of this experiment is reliable although we did not observe a really distinguishable difference between readability of method chains and non-method chains, because $$\Delta \bar{x}_B$$ is a small number in millisecond. To answer RQ 2, we conclude that the non-method chains might be slightly more readable in comparison to method chains. The result of this experiment is almost aligned with a similar study. Börstler and Paech could not observe distinguishable difference between method chains and non-method chains in readability [12].
By considering $z_B = 1.5270$, the result of t-test analysis with

\[
\text{significance level} = 0.05
\]

and two-tailed probability of $\pm z$ yields $t-value = 3.1675$, and the t-test analysis shows that the difference is significant at $p < 0.05$.

People might have different speed of reading. Fast and slow readers might affect the results of experiment B. To study the effect of fast and slow readers on the results of experiment B, we performed an extra analysis to identify the fast and slow readers of this experiment. For more information regarding the process of slow and fast readers’ identification, see appendix D.

By removing the outliers, AFD per method for treatment MCS is 416.2773 ms, and AFD per method for treatment Non-MCS is 408.2381 ms. The results of this study after removing the outliers (fast and slow readers) show that $\bar{x}_{m_{MCS}} > \bar{x}_{m_{Non\,MCS}}$, and it confirms that outliers do not affect the conclusion of experiment B.

### 5.4 Validity Threats

Validity of results is one of the most important concerns in every research. We attempted to minimize these validity threats to this study. These threats to validity are discussed by Wohlin et al. [74]. The validity threats to this study are considered as follows:

#### 5.4.1 Internal Validity

This is a threat to dependent variables. Existence of factors that might affect dependent variables without researchers’ knowledge [73]. This threat had the minimum influence on the results of this study. Confounding factors are the factors may affect dependent variables in an unpredictable way [73]. The influence of confounding factors is usually beyond the knowledge of researchers [73].

Confounding factors that might affect this study are as follows:

- **Mother tongue(s):** Participants who grown up with Latin and non-Latin alphabets may have different speed of source code reading. To lessen the effect of this factor, we defined the participant’s assignment strategy (see section 3.5.1).
• **Glasses and contact lenses:** Glasses and contact lenses may affect the dependent variable FD in an unpredictable way. According to the pilot studies, we observed that the percentage of invalid gaze points increases when the participants wear glasses and contact lenses. To decrease the effect of this threat, we design a restricted approach to validate our datasets. This approach is explained in section 3.9.

• **Head and body movements during the experiment:** Participants may have unintentional head and body movements during the experiment, which can affect the result of eye tracking unexpectedly. We exploited two approaches to overcome this challenge:

  – During the warm-up process, we provide necessary instructions to the participants regarding their head and body movements (see section 3.7.2).
  – We designed a restricted data validation process (see section 3.9).

• **Closed eyes:** Eyes can be closed for short- or long-periods because of the following reasons:
  
  – blinking that is a natural behaviour in human beings and it does not take long-period, and
  – strong light, other irritants, or eye diseases such as dryness may cause the participants to close the eyes for a longer period.

Closed eyes during the experiment may also affect our result unpredictably. To minimize the possible effects of these factors, we designed a restricted data validation process (see section 3.9).

Three groups were studied in both experiments. Participants of each group did not have any interaction with each other. Identity of all participants was kept hidden. Participants of this study were not allowed to attend in more than one experiment, and could only participate in the pilot studies or the main experimentation. The experiment was design to be entirely uncompetitive.

### 5.4.2 Conclusion Validity

This is a threat to the relations between outcomes and the treatments of this study [74]. We can summarize the influential factors increasing the conclusion validity in our study as follows:
• **Proximity of the treatments in experiment A and B:** In MCT and MCS, the numbers of methods are very close. The numbers of methods in MCT and MCS are two and seven, respectively. The number of methods exceeds hardly six or seven methods in a method chain in industry and open source community. One issue was the proximity of these two numbers. Since these two numbers were very close, we could not test method chains with a diverse spectrum of numbers between two and seven. Our pilot study showed that the results are not really distinguishable if numbers of methods are very close. Therefore, we use only method chains with two and seven methods and considering the issues such as construct validity and external validity. If we tested method chains with other number of methods, the result might be indistinguishable.

• **Limitation of accuracy in the eye tracker device:** All devices suffer from some restrictions in a broader sense, hardware, and in a narrower sense, software, if the former is applicable to that device. Tobii® T60 captures data with a frequency of 60 Hz. It means that Tobii® T60 eye tracker captures gaze points 60 times per second. However, Poole et al. stated that reading studies require eye trackers with data rate more than 500 Hz [50]. Therefore, we were considerably constrained by the number of methods that we could test in experiment A. Since the number of methods in treatment MCT is only limited to two methods, the defined AOI for the tests in treatment MCT is very small. Thus, a device with a higher frequency could capture higher number of gaze points.

• **Glasses or contact lenses:** Wearing glasses and contact lenses may lead to inaccurate results. During the data validation process (see section 3.9), we removed the datasets with more than 5% invalid data, and this led to analysis of 18 datasets. To see if glasses and contact lenses can be a serious conclusion validity, we removed datasets with more than 50% invalid data. Therefore, we could increase the number of accepted datasets to 29. The achieved results after considering datasets with more than 50% invalid data resulted in the same conclusion that we draw when we removed invalid datasets with 5% invalid data.

• **Others confounding factors:** Other factors such as mother tongue(s), glasses and contact lenses, head and body movements during the experiment and closed eyes, which are discussed in the section internal validity, can be considered also as the conclusion threats. To avoid repetition, the reasons and the solutions to overcome these threats are discussed in the internal validity section.
5.4.3 Construct Validity

This is a threat to generalization of the study results to concept and theory [74].

- **Interactions among participants**: Participants could not interact with each other, and we kept the identity of all the participants hidden. Each participant could only attend in one of the treatment groups.

- **Number of participants**: Lack of access to an inadequate number of participants to constitute a homogeneous sample, particularly with respect to gender, might be another threats to this study. Number of participants was also an unserious threats regarding the mother tongues of the participants as we applied a participants’ assignment strategy (see section 3.5.1). However, Sharafi et al. examined 37 eye tracking studies in software engineering, and summarized the number of participants for each study [60]. Their results showed the mode number of participants in these studies was 15 and the median was 18 [60]. In our study, the total number of participants were 31, and assessing the validity of the datasets dropped the number of participants (datasets) to 18. This number of participants is equal to the median number of participants in the similar studies.

- **Eye tracking laboratory**: In this experiment, we controlled the eye tracking laboratory environment restrictedly (see section 3.4), and some other factors such as participants sitting posture and other guidelines are used to decrease possible negative effects of the laboratory environment.

- **Eye tracking device**: According to Sharafi et al., Tobii device is used by 18 studies out of 29 similar other studies [60]. In the rest of the studies, FaceLab is used by three studies, Eye-Link II is used by five studies, eye tracker from Applied Science Laboratories is used by two studies, and only one study used EMR-NC [60]. This shows that Tobii device is a common device used for eye tracking studies in software engineering.

- **Measurement**: FD and AFD are used by different researchers to investigate similar studies such as code comprehension [60].

5.4.4 External Validity

This is a threat to generalization of the study results to industry level [74]. This threat had the minimum influence on the results of this study by providing the test materials from open-source community, and good academic
support through studying papers and articles. The results of this study is not only limited to Java programming languages. These results might be used in other programming languages as well.

5.5 Summary

Experiment A provided the answer to RQ 1. Analysis of fixation durations shows that methods with higher number of methods in method chains are slightly more readable than method chains with lower number of methods.

Experiment B provided the answer to RQ 2. Analysis of fixation durations presents that non-method chain style are slightly more readable than method chain style.
Chapter 6

Conclusions and Future Work

The cost of software products may decrease if vital factors affecting the cost of software development are recognized and tackled correctly by researches. Maintenance constitutes merely between 40% and 70% cost of software development life cycle [7, 28, 65]. According to some research, comprehension of source code is a vital cost in maintainability of software applications [6, 29]. Comprehension of codes is affected with readability [11]. It embraces approximately 50% of maintenance cost [25, 46, 75]. Since maintenance causes the code to be modified, readability and understandability should be used as an important measure for the maintainability [1]. Therefore, as the readability of source code increases, the understandability may increase, and cost of software production decreases.

We use an eye tracking device to conduct this study, and investigate programmers’ eye movements with respect to the readability of method chain in Java. We aimed to investigate the existence of relations between readability of Java source codes and the number of methods in method chains; this contributed to the first research question (RQ 1). We designed two different treatments: MCT and MCS to answer RQ 1. Then, we investigated if two different programming styles for methods labeled as method chain and non-method chain vary in level of readability; this formulated the second research question (RQ 2) of this study. To answer RQ 2, we used the results from the treatment MCS, and we also designed a treatment called Non-MCS.

Regarding RQ 1, we found that the number of methods in a method chain has a reverse relation with programmers’ eye fixation durations although the change in number of methods affects the readability of method chain marginally. As the number of methods increases in a method chain, the programmers’ eye fixation duration decreases. That is to say, as the number of methods in a method chain increases, the readability of methods increases. In RQ 2, we discovered that the non-method chain style is slightly more readable than method chain style.
Parallel to the main analysis in this study, we performed two extra analysis: 1. we assessed the effect of 50% invalid data rejection instead of 5% invalid data rejection and also 2. we identified and removed fast and slow readers to see how these outliers affect the results and conclusions of this study. The results of these two extra studies were aligned with the conclusions of this study.

We had different challenges during this study. One major obstacle was the lack of access to an insufficient number of participants, particularly with respect to gender.

Readability is a complex concepts in different fields from psychology to linguistics and from cognitive science to software engineering. It exists in all aspects of our daily life in books, films, advertisements and source codes.

Although research regarding readability in software engineering goes back to 1960’s, the study on readability is quite new in comparison to the other fields.

This study can be investigated deeper, and it also can be extended in order to investigate the difference between novice and professional programmers’ eye movements. Research regarding the readability of source codes can fly beyond the scope of this research and similar research to investigate the readability from programmers’ brain perspective. Research in this field can lead to design of automated application to be able to measure readability of source codes.
References


[43] LORGE, I. Predicting readability. The Teachers College Record 45, 6 (1944), 404–419.

[44] MCCracken, D., AND WEINBERG, G. M. How to write a readable fortran program. Datamation 18, 10 (1972), 73.


Appendix A

Experiment Materials

Figure A.1: Warm-Up 1

How many cats are in the next picture?

Ready?
Appendix A. Experiment Materials

Figure A.2: Warm-Up 2

Task:
Find the word ‘soft’.
Say the line number aloud.

Answer:
Line number 4.

Task:
Find the word ‘fish’.
Say the line number aloud.

Ready?
What is Method Chain?

Task:
Find the method `IfStateProvide()`. Say the line number aloud.

Ready?

Figure A.3: Warm-Up 3
Test 1

Ready?

Task:
Find the method 'oldestWaitStart()'.
Say the line number aloud.

Figure A.4: Test 1, Method chain with two methods.
Appendix A. Experiment Materials

Figure A.5: Test 2, Method chain with two methods.

Test 2

Ready?

Task:
> Find the method `getSelectedItem()`.
> Say the **line number** aloud.
Figure A.6: Test 3, Method chain with two methods.
Appendix A. Experiment Materials

Test 4

Ready?

Task:
Find the method 'parseFromReader()'.
Say the line number aloud.

Figure A.7: Test 4, Method chain with two methods.
Test 5

Ready?

Task:

Find the method `connectToServer()`.
Say the line number aloud.
Test 1

Ready?

Task:
Find the method `getMinimumValue()'.
Say the line number aloud.

Figure A.9: Test 1, Method chain with seven methods.
Test 2

Ready?

Task:
Find the method 'nexueServerHost()'.
Say the line number aloud.
Test 3

Ready?

Task:
Find the method ‘getimageLibrary()’. Say the line number aloud.

Figure A.11: Test 3, Method chain with seven methods.
Appendix A. Experiment Materials

Test 4

Ready?

Task:
Find the method `startDelayedRun()`.
Say the line number aloud.

Figure A.12: Test 4, Method chain with seven methods.
Test 5

Ready?

Task:

Find the method `invalidValueSet()`.
Say the line number aloud.
Appendix A. Experiment Materials

Test 1

Ready?

Task:
Find the method ‘getMinimumValue()’.
Say the line number aloud.

Figure A.14: Test 1, Non-method chain with seven methods.
Figure A.15: Test 2, Non-method chain with seven methods.
Test 3

Ready?

Task:
Find the method `getImageLibrary()`.
Say the line number aloud.

Figure A.16: Test 3, Non-method chain with seven methods.
Test 4

Ready?

Task:
Find the method ‘startDelayedRun()’. Say the line number aloud.

Figure A.17: Test 4, Non-method chain with seven methods.
Test 5

Ready?

Task:
Find the method `invalidValueSet()`.
Say the line number aloud.

Figure A.18: Test 5, Non-method chain with seven methods.
# Appendix B

## Table of Abbreviations

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFD</td>
<td>Average Fixation Durations</td>
</tr>
<tr>
<td>AIFD</td>
<td>Average of Individual Fixation Durations</td>
</tr>
<tr>
<td>AOI</td>
<td>Area of Interest</td>
</tr>
<tr>
<td>ASL</td>
<td>Average of Sentence Length</td>
</tr>
<tr>
<td>AWL</td>
<td>Average of Word Length</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-Specific Language</td>
</tr>
<tr>
<td>FD</td>
<td>Fixation Duration (Dependent variable of this study)</td>
</tr>
<tr>
<td>FRES</td>
<td>Flesch Reading Ease Score</td>
</tr>
<tr>
<td>GPL</td>
<td>General-Purposed Language</td>
</tr>
<tr>
<td>IQR</td>
<td>Interquartile Range</td>
</tr>
<tr>
<td>LTM</td>
<td>Long-Term Memory</td>
</tr>
<tr>
<td>MCS</td>
<td>Method Chains with Seven methods (a treatment in this study)</td>
</tr>
<tr>
<td>MCT</td>
<td>Method Chains with Two methods (a treatment in this study)</td>
</tr>
<tr>
<td>ms</td>
<td>millisecond</td>
</tr>
<tr>
<td>Non-MCS</td>
<td>Non-Method Chains with Seven methods (a treatment in this study)</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>RQ</td>
<td>Research Question</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SRES</td>
<td>Software Readability Ease Score</td>
</tr>
<tr>
<td>STM</td>
<td>Short-Term Memory</td>
</tr>
</tbody>
</table>

Table B.1: Table of Abbreviations.
Appendix C

Mathematical Notations

General:

$\bar{x}_{\text{accepted}}$
The sample mean of the accepted participants’ age.

$\bar{x}_{\text{all}}$
The sample mean of the all participants’ age.

In the Treatment MCT:

$s_{h_{\text{MCT}}}$
The standard deviation of the histogram of participants’ fixation durations.

$\bar{x}_{h_{\text{MCT}}}$
The sample mean of the histogram of participants’ fixation durations.

$\bar{x}_{m_{\text{MCT}}}$
The sample mean of fixation durations per method.

In the Treatment MCS:

$s_{h_{\text{MCS}}}$
The standard deviation of the histogram of participants’ fixation durations.

$\bar{x}_{h_{\text{MCS}}}$
The sample mean of the histogram of participants’ fixation durations.

$\bar{x}_{m_{\text{MCS}}}$
The sample mean of fixation durations per method.
In the Treatment Non-MCS:

$s_{h_{\text{NonMCS}}}$  
The standard deviation of the histogram of participants’ fixation durations.

$\bar{x}_{h_{\text{NonMCS}}}$  
The sample mean of the histogram of participants’ fixation durations.

$\bar{x}_{m_{\text{NonMCS}}}$  
The sample mean of fixation durations per method.
Appendix D

Identifications of Slow and Fast Readers

People might have different speed of reading when they read source codes. Fast and slow readers might affect the results of both experiments A and B. To study the effect of fast and slow readers on the results of experiment A and B, we performed an extra analysis to identify the fast and slow readers of this study. The following steps were performed after the data validation process (see section 3.9).

To remove outliers, we follow two steps: 1. we identified the fast and slow readers, and then 2. we identified the tests with unusual AFD.

Step 1: Identification of fast and slow readers

The following part explains the steps to identify the fast and slow readers in general to eliminate their effects on this study. The reasons that might affect reading speed of source codes are pervasively broad. The speed of reading might be varied if source code readers:

- use Latin or non-Latin alphabets in their mother tongues, or
- suffer from any type of reading disorder.

To identify the fast and slow readers, we follow the steps below:

- The average of fixation durations (AFD) of each participant for all tests of a certain treatment was calculated. This is called *Average Fixation Durations per Participant* ($P_{AFD}$). $P_{AFD}$ is presented with bars in figure D.1.
  For example, in figure D.1, to calculate $P_{AFD}$, we calculated AFD of each participant (P1 to P5) for a particular treatment (e.g. MCT, MCS or Non-MCS).
- The mean of all $P_{AFD}$’s is calculated and presented by orange line in figure D.1.
• One standard deviation of the mean for all $P_{AFD}$’s (for each treatment) is calculated. The gray area in figure D.1 presents one standard deviation of the mean of $P_{AFD}$’s.

• The datasets (participants) with $P_{AFD}$ lower or higher than one standard deviation of the mean of $P_{AFD}$’s were rejected (the red bars in figure D.1).

• The remaining of the participants with normal $P_{AFD}$ (the readers with average of fixation duration within one standard deviation of the mean) are called readers with normal $P_{AFD}$ (the blue bars in figure D.1).

Step 2: Identification of tests with unusually shorter or longer $T_{AFD}$ for readers with normal $P_{AFD}$

To eliminate the tests with unusually shorter or longer $T_{AFD}$, we follow the steps below:

• First, we calculate the average fixation duration (AFD) of each accepted test for a particular participant with normal $P_{AFD}$ in a specific treatment denoted by $T_{AFD}$.

  For instance, from step one, in our example (see figure D.1), only participants P1, P2 and P4 have normal $P_{AFD}$.

  Therefore, if participant P1 has five accepted tests, we calculate $T_{AFD}$ of all tests of participant P1.

• Although all the tests at this step belong to readers with normal $P_{AFD}$ (from the step 1; only P1, P2 and P4), we observed that some tests have still unusually shorter or longer $T_{AFD}$ (presented in red bars in figure D.1).
Figure D.2) than one standard deviation of sample mean of $T_{AFD}$ for a participant with normal $P_{AFD}$. The outlier tests that are out of the gray area are rejected (see figure D.2).

This can have different reasons. The source code readers with normal $P_{AFD}$ might have different speed of reading in few specific tests. These reasons can be enlisted as:

- Existence of distractors, especially in the environment.
- Legibility problems.
- Confusion.
- Thinking.

After rejection of outlier readers and outlier tests, we calculate the AFD per method for each treatment to see how they affect the results or conclusions of this study.