Islamic University of Technology (IUT)

Readability of Code Snippets Included in Stack Overflow Questions: An Exploratory Study

Authors-

Akib Mahmud Rime - 170041002
Mohammed Ashfaq Raiyan - 170041009
Nusrat Jahan Rani - 170041044

Supervisor-

Shohel Ahmed
Assistant Professor
CSE Department, IUT

A thesis submitted in partial fulfillment of the requirements for the degree of B.Sc in Computer Science and Engineering

Academic Year 2020-21

Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
A Subsidiary Organ of Organization of Islamic Cooperation (OIC)
Board Bazar, Gazipur-1704, Bangladesh.
April, 2022.

CERTIFICATE OF APPROVAL

The thesis titled,"**Readability of Code Snippets Included in Stack Overflow Questions: An Exploratory Study**" submitted by Akib Mahmud Rime, Mohammed Ashfaq Raiyan, Nusrat Jahan Rani, St. No. 170041002, 170041009, 170041044 of Academic Year 2020-21 has been found as satisfactory and accepted as partial fulfillment of the requirement for the Degree Bachelor of Science in Computer Science and Engineering on April 25, 2022..

Supervisor:

[illegible]

Shohel Ahmed (Supervisor)
Assistant Professor,
Department of Computer Science and Engineering (CSE),
Islamic University of Technology (IUT), Gazipur.

Declaration of Authorship

This is to certify that the work presented in this report is the result of research and experiments conducted by Akib Mahmud Rime, Mohammed Ashfaq Raiyan, and Nusrat Jahan Rani under the supervision of Shohel Ahmed, Assistant Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is further claimed that neither this study nor any portion of this study has been submitted anywhere else for any other degree or certificate. The text admits information obtained from the published and unpublished work of others, and a list of references is provided.

[illegible]

Akib Mahmud Rime
ID: 170041002,
Computer Science and Engineering department,
Islamic University of Technology (IUT),
Date: 25 April, 2022.

Ashfaq Raiyan

Mohammed Ashfaq Raiyan
ID: 170041009,
Computer Science and Engineering department,
Islamic University of Technology (IUT),
Date: 25 April, 2022.

Nusrat

Nusrat Jahan Rani
ID: 170041044,
Computer Science and Engineering department,
Islamic University of Technology (IUT),
Date: 25 April, 2022.

Dedicated to Almighty Allah
and all who has always been a support

Table of Contents

List of Figures

Acknowledgment

It is high time for us to express our heartfelt appreciation and humble submission to Allah, who blessed us and enabled us to be a part this research. We are very grateful to be under supervision of Shohel Ahmed, Assistant Professor, Department of Computer Science Engineering, IUT. We also like to express our gratitude and admiration to Saikat Mondal, Doctoral Researcher, University of Saskatchewan, Canada. Their inspiration, suggestions, and thoughts have been invaluable for this research. Their proper guidance and suggestions had been a great source of inspiration for us to make this research possible. We are really thankful and greatly indebted to them for their whole-hearted supervision.

Abstract

Stack Overflow (SO) is a popular question and answer site for programmers of all skill levels. In Stack Overflow software developers asks questions with sample code segments, supporting description, and if needed with bug report. However, the quality of the questioner code is poorly readable to attract the solution of the answerer. In this paper, we explore the idea of Java code readability from Stack Overflow question and tried to investigate the impact of the criteria in code readability. This research is the first study on Java code readability from Stack Overflow that we are aware of. We propose a novel dataset of Java code readability criteria from Stack Overflow. Our study consists of 21 readability metric of Java, 241 SO questions, three human annotators, and two experts. With collecting SO Questions with Java tag from SO site, three human annotators manually evaluated the dataset and later the annotation were rechecked by two software experts. Moreover, we tried to analyse the impact of the criteria in code readability and the correlations between readability and different performance measures of SO site like readability vs score mechanism, readability vs accepted answers, and so on. Finally, we discuss how our study will encourage new inquirer to ask a good readable question so that he can get an answer he was looking for as early as possible.

Introduction and Problem Overview

1.1 Overview

Stack Overflow (SO) is one of the largest and most popular online Q&A platforms for developers, where they can exchange knowledge by answering questions and acquire new skills by asking questions from developers and experts. It is most recognized for its public Q&A platform, which is visited by over 100 million individuals each month to ask questions, learn, and share technical skills and knowledge.

One of the most significant features of excellent code is code readability, which, like the art of creating code, is a subjective issue that differs amongst developers.When code is easy to read, it is also straightforward to comprehend, making it simple to debug, maintain, and extend. Writing legible code is easier said than done, and making complex code easy to read and understand is a challenging discipline to master.

Learning to make code legible/understandable is tough for new developers to not just accomplish but also understand since the distinctions between readable and unreadable code can be difficult to measure.

Why Does Code Readability Matter in Stack Overflow?

Software engineers devote a substantial amount of effort on reading source code. If code is not created with readability in mind, the time necessary to maintain it increases. It is critical to evaluate how readable the code is in order to reduce the time required to read and understand it.

When a developer first produces code, their knowledge of the system is very detailed since they have generally worked on the project for some time and have studied the specifications. They understand the code/related code, they understand the system, and everything is in order. No one remembers the system after 6 months or a year, thus one must study the code to understand how it works. Poorly written code may make this extremely difficult; code with poor readability is difficult to comprehend, takes longer to debug, is difficult to maintain, and is difficult to extend.

1.2 Problem Statement

Previous studies on code readability discusses the correlation of code readability with quality of code, understandability of the code, cognitive pressure of the reader, question answering timespan, and detection of bug in code. The studies also discusses about Java more readable than any other languages. So, a good code has characteristic like self-explanatory, easy to understand, less error-prone, and less reading pressure. But, there is lack of dataset on code readability of Stack Overflow and corresponding work on it. So, we propose a novel dataset on questions' code readability of Java code in Stack Overflow. We believe that an enriched dataset for code readability of Stack Overflow can enhance the quality of the question.

1.3 Motivation and Research Scope

Code readability is an important part of question in Stack Overflow site. A good code can bring an answer in short time [1, 2]. In paper [2] they found that low readable code are not self-explanatory to a developer. A badly written code, on the other hand, will distract the person replying and Allow the question to be left unsolved. In Stack Overflow a user can ask question, answer a question, can do comment on question-answer and edit question-answer. A question gets a good score when it is found useful by Stack Overflow user [3]. In [4] they worked on 120 github projects finding that code readability has impact on project time and software design quality. In paper [5], their discovery is that the more readable the code is, the more cognitive load on the developer is reduced, but poorly readable code increases it. In paper [6], their finding is the less the code is readable the more it will be error-prone. In paper [7], they have analysed that

Java is more readable than C#. In paper [8] their conclusion is that eliminating nesting enhances reading, and that readability also depends on the readers' grasp of English. In paper [9], they have conducted study on Java coding standards and found that strongest code constructs namely, presence of comments, spacing, while loop, meaningful names, and do-while loop, positively impact code readability. Furthermore, there is a lack of a new dataset on Stack Overflow Java Code Readability and a work on it. So, evaluating coding metrics of popular language Java and maintaining a novel dataset will be a great contribution in the upcoming research scope and to the development community.

Related Works

There has been several investigations from different perspectives on code readability.To the best of our knowledge, ours is the first study that evaluates the readability of Java code in Stack Overflow queries. But there are a few works that analyzes the code readability for different languages in other platforms.

Duijn et al. [1]presented an approach to improve the classification of high and low quality questions based on a novel source of information: the analysis of the code metric in SO questions. For classification they have applied machine learning algorithms to classify a question as high and low quality on basis of analysed code metric.

Treude et al. [2] analyzes a survey-base study to investigate how understandable the Stack Overflow code is to the developer. They conducted a survey consisting of 321 participants. They found that less readable code, incomplete fragments of the code, bad code organization, bad naming issues make the code difficult to understand for the developers.

Umme Ayda et. al [4] aimed to see the average readability of Open Source Software (OSS) projects and how they impact time and software design quality. They worked on 120 projects from Github where they observed that the OSS projects have high readability scores and maintain high scores over time; readability score is measured by code smells and does not correlate with design quality. The limitation of this study is that their findings are limited to OSS projects from Github only.

Fakhoury et al. [5] examined 548 instances of source code from 63 engineered Java programs. To accomplish so, they conducted an investigative study on 3 state-of-the-art models. The investigation was about the models could identify the improving features from source code or not. However, the readability models fail to identify the improving features. Therefore, the readability models could not be used to day to day code maintenance task. As a solution, new readability tools may be developed while keeping the overall readability metrics in mind.

Buse et al. [6] correlated the software quality such as finding bugs etc. with code readability metrics like line of length, identifiers, identifier length, indentation, keywords numbers, spaces, parenthesis, blank lines, comments, periods, commas, and occurrences of any single character and stated that the less the code is readable the more it will be error-prone.

In paper [7], Batool et al. aimed to find programming criteria (features) and their impact on source code readability comparing Java and C from which they found that Java is a more readable language than C#. They used metrics like- ARI, SMOG, Gunning Fog Index, FKR, and CLI and selected 22 criteria that might affect code readability additionally Inheritance, Overriding, Recursive, Nested If, Comments, and Scope than [10]. They also compared the results of the expert evaluation from 15 experts with automated Readability Index Grade level.

Johnson et al. [8] presented an empirical study on two code constructs namely, nesting and looping where the main analysis was that minimizing deep nesting improves readability and also there is no drastic improvement in readability in exchanging a do-while loop with a while loop with an additional observation from the study is that readability depends on readers' knowledge of English.

Duaa Alawad et al. [9] presented an empirical study of the relationships between code readability and program complexity analyzing 35 Java programs with 23 Java coding standards. The analysis consisted of six readability metrics and two metrics with complexity. The six readability metrics are namely, Flesch-Kincaid Readability Index (FKI), The Gunning's Fog Index (GFI), The Simple Measure of Gobbledygook (SMOG), Automated Readability Index (ARI), Coleman-Liau Index (CLI), and Buse Readability Score (BRS). Halstead Complexity Volume and McCabe's Cyclomatic Complexity are the two complexity metrics they utilized. The analysis empirically showed negatively correlation between code readability and program complexity. After empirically investigating the correlation, they applied a machine learning technique, to identify code constructs that substantially affect code readability. The code constructs discussed in the paper are namely, Lines of Code (LOC), Meaningful Names, Comment Indents, Indents, Scope, Inheritance, Polymorphism, Class Distribution, Spacing, Recursive, Formulas, Consistency, Line Length distribution, identifier name Length, Identifier frequency, IF-else, Nested if, For Loop, While Loop, Do-While Loop, Nested -loop, Switch, and Array. According to analysis, the five strongest code constructs that positively affect code readability are namely, the presence of comments, spacing, while loop, meaningful names, and do-while loop.

Tariq et al. [10] conducted a comparative study on the readability of high-level languages- C++, C, and Python comparing human judgment with readability indexes like- ARI, SMOG, FOG, and FKG and found that Java is more readable than the other two. They chose 14 parameters to measure code readability - Parenthesis, Indentation, Spaces, Loops, and so on. In their study, they took an expert survey and then again computed readability using a source code readability tool (SCRT). They also analyzed their results statistically using the SPSS tool to verify the effectiveness of the experiments.

Authors of paper [11] presented an empirical study of the role played by textual features in code readability evaluating 600 code snippets manually by more than 5000 people, based on which they proposed a set of textual features to improve state-of-the-art models. correlated. Comments and Identifiers Consistency (CIC), Identifier Terms in Dictionary (ITID), Narrow Meaning Identifiers (NMI), Comments Readability (CR), Textual Coherence (TC) and Number of Meanings (NM) are the textual features.

Scalabrino et al. [12] proposed two new textual features based on lexical analysis of manually evaluated source code snippets and tried to establish correlation among textual-based features, structural features, and code readability. They showed that the combination (i.e., structural+textual) improves the accuracy of code readability models. Moreover, they replicated a study by Buse and Weimer [6] on the correlation between readability and FindBugs warnings.Their study showed that unreadable codes are more close to having issues.

Dos et al. [13] described a survey that assessed the impact of a set of Java coding practices on the readability perceived by software developers.

Fakhoury et al. [14] offered a research that linked code readability and cognitive load in another work.Study shows that participants with lower cognintive load faced high readable code and people with higher cognitive load faced low readable code.

Tashtoush et al. [15] developed an automatic evaluation tool to evaluate impact of programming features on code readability. And to assess the impact properly, a manual survey was done. The survey consists of 23 questions for 23 features which was evaluated by 141 random annotators or experts several software developing companies. They used features namely, Arithmatic formulas, Nested loop, Recursive functions, Line Length distribution, overriding, Identifier frequency, and so on. The survey results were analyzed by a tool. According to the studies, meaningful names, comments, and consistency improve code readability. Recursive functions, nested loops, and arithmetic formulae, on the other hand, have a detrimental influence on source code readability. And other features had no impact.

Raymond et al. [16] conducted an correlation analysis between automatic analysis and manual analysis over 100 code snippets. The manual analysis was conducted by the judgements of 120 human annotators. Then, correlated the automatic analysis with manual analysis and findings was 80

Study Methodology

3.1 Dataset Preparation

Figure 3.1 represents the data collection steps. We collect the data from data dump of Stack Overflow Stack Exchange Data Explorer [17]. Particularly, we collected questions with Java tag from the data dump since Java is a very popular programming language in the software industry and also because Java is popular topic in the SO site. We collected around 450 questions with <java> tag from the data dump.

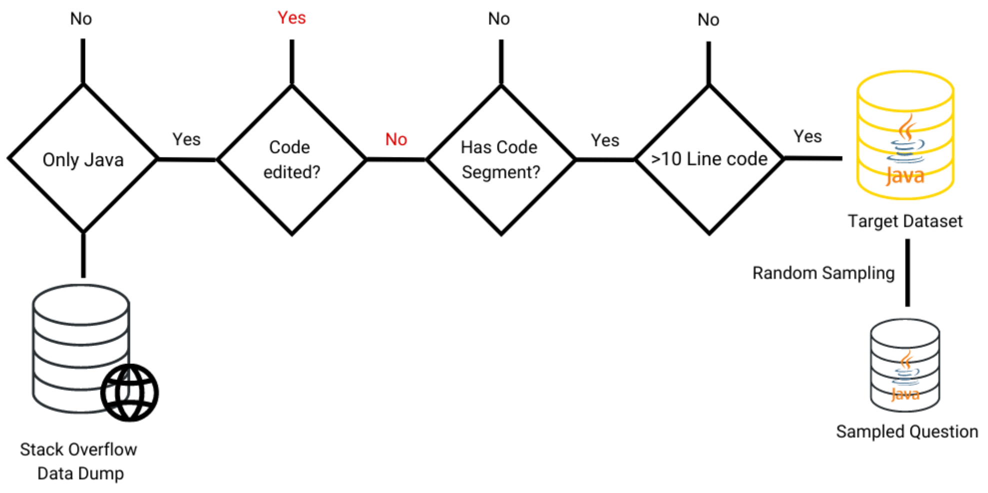

Figure 3.1: Dataset Preparation

Especially, we preferred to select the non-edited version of the question to get the raw version of code the author asked so that we can get codes with readability issues and filtered the question accordingly. Next to achieve our target dataset, we imposed a restrictions on the collected

data if the dataset contains code section or not. Next, we imposed another restrictions on our dataset by checking if code section has greater than 10 lines of code because we had to ensure the code is standard to evaluate against 21 code metric. If the code snippet is too small it might miss lots of criteria those we considered. According to our investigation, we finally got 241 question out of 450 question(i.e.,53.5%) as our final dataset after evaluation which have at least greater than 10 lines of Java code.
During manual analysis we evaluated the dataset against 21 coding metric, we named the metrics as "Criteria". Each criterion was evaluated on scale of 3 parameters namely, -1 if the criteria is not properly written according to guidelines, 0 if the criteria is absent in the code section, +1 if the criteria is well written.
The dataset was evaluated separately by 3 annotators and the annotation was reviewed again to maintain consistency in the evaluation. Later on, the annotated dataset was reviewed by 2 experts.

3.2 Criteria

What is Criteria and why it is important
In case of code readability, we can judge the code from various angles. Code smell is a very important part to notice in case of code readability. When any selected criterion is violated then it's the weakness of the code. If this weakness is ignored then this will lead to poor code readability.
We all need to write code which will be scalable and will be used for long time. So, if these criteria are maintained then code quality will increase eventually. The code will be more readable and in the future different people will understand the code easily which is really necessary for a long term solution.

3.2.1 MEANINGFUL NAMING OF THE IDENTIFIERS

Naming convention plays a vital role in case of Java code readability. Readability of Java code is important because it means less time is spent trying to figure out what the code does, leaving more time to fix or modify it. Different languages follow different conventions for **identifier** naming. While coding if the convention is not followed then it creates code smell which is very bad in terms of code readability. In Java, different identifiers follow different naming convention.

So, the first and foremost concept of writing the identifiers is to make those meaningful (even if the identifier name gets longer)

VARIABLE NAMING

There are many naming conventions in case of variable declaration.

Multi-word Delimited

This convention is to separate words in a variable name without the use of white-space. White-space within variables is usually difficult for programming languages to interpret. Because of this variables must be delimited in some way. Here are several delimiting conventions commonly used in code:

- **Snake-case**: Words are delimited by an underscore.
- **Pascal-case or Upper Camel-case**: Words are delimited by capital letters.
- **Lower Camel-case**: Words are delimited by capital letters without the very first letter.
- **Hungarian Notation**: This notation describes the variable type or purpose at the start of the variable name, followed by a descriptor that indicates the variable's function. The Camel-case notation is used to delimit words.

In Java, the convention is to use the Camel-case style for the regular variable naming.

```java
public class Test {
    public String firstName;
    public int age;
}
```

Listing 3.1: Variable Naming

METHOD NAMING

Method naming follows the same convention as the variable naming of Java.

CLASS NAMING

In case of class naming we use the Upper Camel Case or Pascal case naming convention.

```java
public class Test {

}
```

Listing 3.2: Class Naming

CONSTANT NAMING

Java constants should be all UPPERCASE where words are separated by underscore character ("_"). Make sure to use the final modifier with constant variables.

```java
public class Test {
    public final int MAX_LIMIT = 256;
    public final int TOTAL_DAYS_COUNT = 30;
}
```

Listing 3.3: Constant Naming

3.2.2 METHOD PARAMETERS

We use the number of method params as a criteria for java code readability. For our manual analysis we used 3 as the upper limit till where a regular code reader feels easy to read the method. Sometimes, it's necessary to pass more than 3 params in a method but it always slows down the code readability.

3.2.3 LIMIT LINE LENGTH

In Stack Overflow, we have a very short box to show our code. So, when the code is longer then we need scrolling to see the right part of the code snippet. So, this is a criteria to check wheather each line of the code snippet fits the box or we need scrolling. We know that the standard line limit for the popular languages is within range 80–120. But in case of StackOverflow we just don't want the code submitters to paste such a code which needs scrolling to the right.

```
public class MyDataService {

    private SqlMapClient sqlMap;

    @Autowired
    public MyDataService (SqlMapClient sqlMap) {
        this.sqlMap = sqlMap;
    }

    @Transactional(readOnly = true)
    public MyData getSomeData() {
        // an instance of sqlMap connected to slave should be used
    }

    @Transactional(readOnly = false)
    public void saveMyData(MyData myData) {
        // an instance of sqlMap connected to master should be used
    }
}
```

Listing 3.4: Accepted Criteria Example for limit line length [18]

```
public EntryPoint() {
      try {
           HttpServletRequest request = (HttpServletRequest) FacesContext
    .getCurrentInstance().getExternalContext().getRequest();
           HttpServletResponse response = (HttpServletResponse)
    FacesContext.getCurrentInstance().getExternalContext().getResponse
    ();
           String loginID = request.getParameter("loginID");
           //Do some code to load the user/permissions
           response.sendRedirect(
              //The appropriate page
           );
```

```java
      } catch (IOException ex) {
         logger.error(null, ex);
      } catch (SQLException ex) {
         logger.error(null, ex);
      }
   }
```

Listing 3.5: Rejected Criteria Example for limit line length [19]

3.2.4 CODE FORMATTING

Code formatting is one of the major criteria for the code readability. Actually there are different code formatting style which are followed by developers. The initial look of the code is the code formatting. If someone who wants to answer any question in SO and see a code snippet which is not formatted in a way that he/she is familiar with then it's a very big readability issue.

So, in this criteria we focused on the code formatting style which are conventional in Java and we penalized if this proper indentation style is not followed.

In Java, two widely used formatting styles are ***ALLMAN and K&R***. So, we only allowed these two criteria and deviation from this is always penalized.

```java
public class MyDataService {

    private SqlMapClient sqlMap;

    @Autowired
    public MyDataService (SqlMapClient sqlMap) {
        this.sqlMap = sqlMap;
    }

    @Transactional(readOnly = true)
    public MyData getSomeData() {
        // an instance of sqlMap connected to slave should be used
    }

    @Transactional(readOnly = false)
    public void saveMyData(MyData myData) {
        // an instance of sqlMap connected to master should be used
    }
}
```

Listing 3.6: Accepted Criteria Example for code formatting [20]

```
import java.util.Arrays;
import java.util.Collections;

public class CodeTestingClass
{

    public static void main(String[] args)
    {

     Integer[] array = {1,2,3,4,5};

     Collections.rotate(Arrays.asList(array), 1);

     System.out.println(Arrays.toString(array) + "\n" );

  for(Integer i : array)
  {

   System.out.print(i);

  }

  System.out.print("\n");

  for(int i : array)
  {

   System.out.print(i);

  }

    }
}
```

Listing 3.7: Rejected Criteria Example for code formatting [21]

3.2.5 CONSISTENCY

This criteria is concerned about the consistency of the code snippet from different angles. Code formatting consistency, spacing consistency, naming consistency, code grouping consistency are taken care of in this criteria. Usually, experienced developer's code are always consistent in terms of readability. We evaluated every code snippet in our dataset using the subcriterias of the consistency. Since we focused on many sub-criterias we marked the code snippet bad only when found serious consistency issues where is code was inconsistent in many major portions. This inconsistency in the code is very irritating for the code readability.

```java
Log.i("GAME.DrawThread", "run()");
Log.i("GAME.DrawThread", Thread.currentThread().getName());
Canvas canvas = null;
try {
    canvas = holder.lockCanvas();
    synchronized(holder) {
        Log.i("GAME", "draw():synchronized");
        Paint paint = new Paint();
        paint.setColor(R.color.draw_color);
        canvas.drawColor(R.color.draw_color);
        canvas.drawLine(0, 0, 500, 500, paint);
    }
} catch (SurfaceHolder.BadSurfaceTypeException e) {
    Log.e("GAME", "onDraw():  BadSurfaceTypeException");
} finally {
    if (canvas != null) {
        holder.unlockCanvasAndPost(canvas);
    }
}
```

Listing 3.8: Accepted Criteria Example for consistency [22]

```java
Log.i("GAME.DrawThread", "run()");
Log.i("GAME.DrawThread", Thread.currentThread().getName());
Canvas canvas = null;
    try {
      canvas=holder.lockCanvas();
        synchronized(holder) {
            Log.i("GAME", "draw():synchronized");
            Paint paint = new Paint();
            paint.setColor(R.color.draw_color);
            canvas.drawColor(R.color.draw_color);
            canvas.drawLine(0, 0, 500, 500, paint);
        }
    } catch (SurfaceHolder.BadSurfaceTypeException e) {
    Log.e("GAME", "onDraw():  BadSurfaceTypeException");
} finally {
    if (canvas != null)
    {
        holder.unlockCanvasAndPost(canvas);
    }
}
```

Listing 3.9: Rejected Criteria Example for consistency [23]

We see that we have indentation and formatting inconsistency in the code snippet.

3.2.6 SIMPLICITY

Simplicity is the part where the code is well organized, easier to understand. Some code snippets are always bad because of the algorithmic poor thinking. Sometimes it is important to add extra methods to understand the flow of the algorithm easily. This approach can even make a complex algorithm simple in terms of understanding.

Generally, experience programmers code are simpler. They focus on the flow of algorithm. While they code they constantly refactor. For example, whenever an experienced programmer smells duplication he/she makes a function to reuse it for better readability.

So, we evaluated every code snippet in our dataset and checked whether the current code could be made simpler. We marked the code snippet bad in terms of simplicity when we see major simplicity problem which makes the code harder to read or which increases the understanding time due to the unnecessary complexity of the snippet.

```
public class Test {
    public static void main(String[] args) {
        int first = 2, second = 3;
        int temp = first;
        first = second;
        second = temp;
        System.out.println(first + " " + second);
    }
}
```

Listing 3.10: Accepted Criteria Example for simplicity

```
public class Test {
```

```java
    public static void main(String[] args) {
        int first = 2, second = 3;
        first ^= second;
        second ^= first;
        first ^= second;
        System.out.println(first + " " + second);
    }
}
```

Listing 3.11: Rejected Criteria Example for simplicity

Simple **Swaping Algrithm** can be made complex degrading the code readability.

3.2.7 CLARITY

In terms of code readability, when we go through a code snippet, in most of the codes we have functions and variables which are named by the programmer. The criteria clarity says about the actual implementation of the function. The code snippet is unclear when the naming doesn't match will the method code. In case of variable naming we also check the clarity of the flow of the variable. It's hard to understand when the use of variable says something different than that of the name.

In our manual evaluation we checked if the code snippets violate this criteria. This criteria is very important because even if the code seems clear and nice this issue will decrease the code quality and readability. It will take longer time to understand the code if the unclear function is used many times in the code snippet.

```java
public class Test {
    public static void main(String[] args) {
    }

    public int addTwoNumbers(int a, int b) {
        int result = a + b;
        return result;
    }

    public void printResults(int result) {
        System.out.println("The result is " + result);
    }
}
```

Listing 3.12: Accepted Criteria Example for clarity

```
public int addTwoNumbers(int a, int b) {
    result = a + b;
    System.out.println("The result is " + result);
    return result;
}
```

Listing 3.13: Rejected Criteria Example for clarity

Here, the function should only add two numbers but it's **adding** and **printing** which is not clear in the method name.

3.2.8 CODE GROUPING

Code grouping is to group code portions so that each portion has a meaning or refers to part of algorithm. When the code grows code grouping is a must. Without fulfilling this criteria it's sometimes impossible to keep track while reading.

Experienced programmers group code while coding. The mostly used common rule for code grouping is to add extra-line before the loops and methods. They group the variable declaration part and loop part differently to make it more readable.

So, based on this criteria we evaluated our code snippets and marked bad when we found the programmer not to group where it was necessary. While reading code, it's easier to hold focus on the code if the necessary grouping is done and documented properly.

```
import java.util.*;

public class Test {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int t = Integer.parseInt(in.nextLine());

        while (t-- != 0) {
            int n = Integer.parseInt(in.nextLine());
            String s[] = in.nextLine().split(" ");
            int a[] = new int[n];

            for (int i = 0; i < n; i++) {
                a[i] = Integer.parseInt(s[i]);
            }

            int d = 0;

            for (int i = 0; i < n - 1; i++) {
                d += (a[i + 1] - a[i] - 1);
            }

            System.out.println((d <= 2) ? "YES" : "NO");
        }
    }
}
```

Listing 3.14: Accepted Criteria Example for code grouping

```
import java.util.*;
public class Main{
    public static void main(String[] args){
        Scanner in=new Scanner(System.in);
        int t=Integer.parseInt(in.nextLine());
        while(t--!=0){
            int n=Integer.parseInt(in.nextLine());
            String s[]=in.nextLine().split(" ");
            int a[]=new int[n];
            for(int i=0;i<n;i++) a[i]=Integer.parseInt(s[i]);
            int d=0;
            for(int i=0;i<n-1;i++) d+=(a[i+1]-a[i]-1);
            System.out.println((d<=2)?"YES":"NO");
        }
    }
}
```

Listing 3.15: Rejected Criteria Example for code grouping

3.2.9 AVOID DEEP NESTING

Deep nesting is very irritating for code readability since while reading inner loop we need to keep in mind how the outer loops affect the inner loop. Since nesting is the part of code we can't ignore that. But there are technique to avoid deep nesting in some situations. Experience programmers try their best to avoid deep nesting using simple hacks wherever possible.

We checked for the nested loops and if found then we penalized for that in terms of readability.

```
public static void do_stuff() {
    if (!is_writable($folder)) {
        return false;
    }

    if (!$fp = fopen($file_path,'w')) {
        return false;
    }

    if (!$stuff = get_some_stuff()) {
        return false;
    }

    if (fwrite($fp,$stuff)) {
        // ...
    } else {
        return false;
    }
}
```

Listing 3.16: Accepted Criteria Example for avoiding deep nesting

```
public static void do_stuff() {
    if (is_writable($folder)) {

        if ($fp = fopen($file_path,'w')) {

            if ($stuff = get_some_stuff()) {

                if (fwrite($fp,$stuff)) {

                    // ...

                } else {
                    return false;
                }
            } else {
                return false;
            }
        } else {
            return false;
        }
    } else {
        return false;
    }
}
```

Listing 3.17: Rejected Criteria Example for avoiding deep nesting

3.2.10 IMPLEMENTING METHODS

Better method implementation is necessary for readability. Sometimes in the subroutine of the method we see early return for different cases which make code more readable since other part of subroutine can be ignored for the edges cases. Returning **null** is also discouraged.

While evaluating methods we carefully took a look on what not to return, method parameters, fail fast & return early, conditionals, apply Single responsibility principle (SRP) etc. These SRP and other issues are very important since violation of these will be a big distraction in terms of code readability.

```java
public int linearSearchPos(List<Integer> arr, int value) {
    for (int i = 0; i < arr.size(); i++) {
        if (arr.get(i) == value)
            return i;
    }
    return -1;
}
```

Listing 3.18: Accepted Criteria Example for method implementation

```java
public Integer add(int a, int b) {
    return null;
}
```

Listing 3.19: Rejected Criteria Example for method implementation

3.2.11 DRY PRINCIPLE

This is one of the most important criteria in software development and programming. "Don't repeat yourself" (DRY) is a principle of software development aimed at reducing repetition of software patterns,[1] replacing it with abstractions or using data normalization to avoid redundancy. The DRY principle is stated as "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system" Usually this is an intense readability issue when same code or subroutine algorithm is repeated. This coding style is highly discouraged. So, while evaluating we checked these repeat and penalized if DRY PRINCIPLE is not followed.

```java
public StringBuilder getProcessString(StringBuilder str, int maxLength)
    {
    StringBuilder now = new StringBuilder();
    int extra = (maxLength - str.length()) / 2;
    for (int k = 0; k < extra; k++) {
        now.append(" ");
    }
    return now;
}

public List<String> getAllCenterAlignedLines(String text, int maxLength
   ) {
    StringBuilder so_far = new StringBuilder();
    List<String> all = new LinkedList<>();
    int n = text.length();
    for (int i = 0; i < n - 1; i++) {
        if (text.charAt(i) == '\r' && text.charAt(i + 1) == '\n') {
            all.add(getProcessString(so_far, maxLength).toString());
            so_far = new StringBuilder();
        } else {
            so_far.append(text.charAt(i));
        }
    }
    // Now we have the last line
    all.add(getProcessString(so_far, maxLength).toString());
    return all;
}
```

Listing 3.20: Accepted Criteria Example for dry principle

```java
public List<String> getAllCenterAlignedLines(String text, int maxLength
   ) {
    StringBuilder so_far = new StringBuilder();
    List<String> all = new LinkedList<>();
    int n = text.length();
    for (int i = 0; i < n - 1; i++) {
        if (text.charAt(i) == '\r' && text.charAt(i + 1) == '\n') {
            StringBuilder now = new StringBuilder();
            int extra = (maxLength - so_far.length()) / 2;
            for (int k = 0; k < extra; k++) {
                now.append(" ");
            }
            now.append(so_far);
            all.add(now.toString());
            so_far = new StringBuilder();
        } else {
            so_far.append(text.charAt(i));
        }
    }
    // Now we have the last line
    StringBuilder now = new StringBuilder();
    int extra = (maxLength - so_far.length()) / 2;
    for (int k = 0; k < extra; k++) {
        now.append(" ");
    }
    now.append(so_far);
    all.add(now.toString());
    so_far = new StringBuilder();
    return all;
}
```

Listing 3.21: Rejected Criteria Example for dry principle

3.2.12 CONSISTENT TEMPORARY NAMES

Temporary variables are often used in the code snippet of StackOverflow. A temporary name is a variable with short lifetime, usually to hold data that will soon be discarded, or before it can be placed at a more permanent memory location. Because it is short-lived, it is usually declared as a local variable, i.e. a variable with local scope.

In this criteria, we evaluated the for loops where temporary variable is used. Iterator i and j may not be that meaningful but it is used conventionally. We checked this criteria while evaluating code snippets. [24]

```
public static void main(String[] args) {
    for (int i = 0; i < array1.size(); i++) {
        for (int j = 0; j < array2.size(); j++) {
            // code goes here ....
        }
    }
}
```

Listing 3.22: Accepted Criteria Example for consistent temporary names

```
public static void main(String[] args) {
    for (int i = 0; i < array1.size(); i++) {
        for (int element = 0; element < array2.size(); element++) {
            // code goes here ....
        }
    }
}
```

Listing 3.23: Rejected Criteria Example for consistent temporary names

3.2.13 AVOID USING MAGIC NUMBERS

Magic numbers are those numbers which we suddenly discover inside our code without any meaning. Many code snippets contains random index accessing and random numbers are passed as a parameter without any trace before. Experienced programmer avoid these magic numbers wherever they can. It's always better to declare a meaningful variable for the magic number and use the variable. This make the code more readable.

We evaluated our code snippets according to this magic number avoiding criteria. We didn't penalize the obvious numbering. For example, arr[0] means the first element of the array which is trivial.

Magic number avoiding is necessary because this numbers adds extra complexity to understand and hence code readability gets affected highly.

```
public class Foo {
    public static final int MAX_PASSWORD_SIZE = 7;

    public void setPassword(String password) {
         if (password.length() > MAX_PASSWORD_SIZE) {
              throw new InvalidArgumentException("password");
         }
    }
}
```

Listing 3.24: Accepted Criteria Example for magic numbers

```
public class Foo {
    public void setPassword(String password) {
         // don't do this
         if (password.length() > 7) {
              throw new InvalidArgumentException("password");
         }
    }
}
```

Listing 3.25: Rejected Criteria Example for magic numbers

3.2.14 NOISY CODE

Noisy codes are those codes which are useless and redundant. There are portions of the code which creates extra noise and hinders readability. This noise can be made by unnecessary code comments, a declared variable which is never used.

Generally, programmers should be very careful about the extra noise of the code and constant refactoring reduces this noise and make it cleaner and increase readability.

So, while evaluating the code snippets we checked the dead codes and comments which causes noise and marked accordingly.

Noise inside the code should always be prevented since this extra unnecessary information causes readability issue.

```java
public class Foo {
    public static void main(String[] args) {
        String str = "main";
        String [] array = {"main"};
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);
        foo(str, array);
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);
    }

    static void foo(String str, String[] array){
        str = "foo";
        array[0] = "foo";
    }
}
```

Listing 3.26: Accepted Criteria Example for noisy code

```java
public class Foo {
    public static void main(String[] args) {
        //declaring and initializing String variable 'str' with value "
    outside"
        String str = "main";

        //declaring and initializing Array 'array' with values
        String [] array = {"main"};

        //printing values of str and array[0]
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);

        //calling function foo()
        foo(str, array);

        //printing values after calling function foo()
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);
    }

    static void foo(String str, String[] array){
        str = "foo";
        array[0] = "foo";
    }

}
```

Listing 3.27: Rejected Criteria Example for noisy code

3.2.15 KNOWLEDGE OF ENGLISH LANGUAGE

This criterion is about the reading ease of the code. It's necessary to evaluate if the code elements use proper English terms (where applicable) and if they are easy. We checked the level of English language from the perspective of school level, college level and university level vocabulary understanding.

Experience programmer always use easier and meaningful words while coding which eventually increases code readability. If it's possible then choosing the school level English wording is always recommended.

So, for this criterion we evaluated if the wording is of school or college level. Choosing the dictionary vocabulary which are hard to understand for school or college level students are penalized.

While coding it's necessary to remember that the code has to be easy to read and understand. If the code cuts time to understand the meaning of the English word then it's very bad in terms of code readability.

```java
public class MyDataService {
    private SqlMapClient sqlMap;

    @Autowired
    public MyDataService (SqlMapClient sqlMap) {
        this.sqlMap = sqlMap;
    }

    @Transactional(readOnly = true)
    public MyData getSomeData() {
        // an instance of sqlMap connected to slave should be used
    }

    @Transactional(readOnly = false)
    public void saveMyData(MyData myData) {
        // an instance of sqlMap connected to master should be used
    }
}
```

Listing 3.28: Accepted Criteria Example for the knowledge of English Language

```java
public static void main(String[] args) {
    String sentence1 = "The cat ate the fish";
    String[] s1Split = sentence1.split(" ");

    String wnhome = "C:/Program Files/WordNet/2.1";
    String path = wnhome + File.separator + "dict";
    URL url = new URL("file", null , path);
    IDictionary dict = new Dictionary(url);
    dict.open();
}
```

Listing 3.29: Rejected Criteria Example for the knowledge of English Language

3.2.16 CODE COMMENT

Code comment plays a vital role while reading code snippet and programmers use it on a regular basis. But commenting is also a skill which can enhance code readability. We can have two types of commenting. One is Good Commenting and other is Bad commenting. It is highly discouraged to put redundant comments if a method or a variable says what it does (Obvious comments). While evaluating if we find any commented out code then we always penalized for that. Since these are Java code snippets we allowed JavaDoc comments as necessary.

Sometimes, commenting are dangerous because at the time of refactoring, programmers frequently forget to update their previous comment.

That's why it's always better to avoid commenting by writing the variable or method name nicely.

```java
protected void onStart() {
    super.onStart();

    if(isFirstLaunch()){
        populateDefaultQuotes();

        //Save the preferences, isFirstLaunch will now be false
        SharedPreferences settings = getSharedPreferences(Constants.
    PREFS_NAME, 0);
        SharedPreferences.Editor editor = settings.edit();
        editor.putBoolean("isFirstLaunch", false);
        editor.commit();
    }

    setupUI();
    checkOrientation();
    restoreCache();
}

private void populateDefaultQuotes(){
    System.out.println("!!!!!! FIRST TIMER !!!!!!");
}

private boolean isFirstLaunch() {
    // Restore preferences
    SharedPreferences settings = getSharedPreferences(Constants.
    PREFS_NAME, 0);
    boolean isFirstLaunch = settings.getBoolean("isFirstLaunch", false)
    ;

    return isFirstLaunch;
}
```

Listing 3.30: Accepted Criteria Example for code comment

```java
public class Foo {
    public static void main(String[] args) {
        //declaring and initializing String variable 'str' with value "
    outside"
        String str = "main";

        //declaring and initializing Array 'array' with values
        String [] array = {"main"};

        //printing values of str and array[0]
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);

        //calling function foo()
        foo(str, array);

        // foo(str, array);
        // foo(str, array);

        //printing values after calling function foo()
        System.out.println("str : " + str + " , array[0] : " + array
    [0]);
    }

    static void foo(String str, String[] array){
        str = "foo";
        array[0] = "foo";
    }
}
```

Listing 3.31: Rejected Criteria Example for code comment

3.2.17 CLASS ORGANIZATION

Class organization is a criterion where the coupling and cohesion among classes are observed keenly. Experience programmers follow the SOLID principles and design patterns properly and their classes remain well organized. In a class, we should only put methods that are specific to the particular class. And it's a good practice to always try to code in interface.

While evaluating the Java code snippets, we have checked the relations of variables and methods to their class. We penalized when we see unnecessary methods are within the classes and the cohesion and coupling rules are not followed.

In SO code snippets, Class Organization is important when people share their part of projects and ask questions from there. So, to answer that question, the underlying structure and relations of the classes have to be understood. Here, Good Class Organization always increases the code readability.

```
public Class File {
    private boolean hidden;
```

```java
    private boolean read;
    private boolean write;

    public boolean isHidden() {
        return hidden;
    }
    public void setHidden(boolean hidden) {
        this.hidden = hidden;
    }
    public boolean isRead() {
        return read;
    }
    public void setRead(boolean read) {
        this.read = read;
    }
    public boolean isWrite() {
        return write;
    }
    public void setWrite(boolean write) {
        this.write = write;
    }
}
```

Listing 3.32: Accepted Criteria Example for class organization

```java
public class Addition {
    public int add(int a, int b) {
        return a + b;
    }

    public int mul(int a, int b) {
        return a * b;
    }

    public void print(int a, int b) {
        System.out.println(add(a, b) + " " + mul(a, b));
    }
}
```

Listing 3.33: Rejected Criteria Example for class organization

3.2.18 HOW MUCH DESCRIPTION SUPPORT CODE READABILITY

In SO, short code snippets are posted and asked questions on that to get the answer. So, each question has descriptions regarding the problem. Sometimes, it is mandatory to read the description to understand the problem and corresponding code snippet. In this criterion, we checked the effect of good description to enhance code readability. It's necessary to describe the code snippet to the point. The quality of description degrades when it's hard to find relation with the problem. Sometimes, it's better to describe part of the ambiguous code since the answerer will not get the whole project to understand.

3.3 Qualitative Analysis

We, three of the authors, participated in the manual analysis and spent a total of 100 man-hours on the 400 questions, from which we filtered out 241 data that matched all of our constraints and criteria to be evaluated.

We followed a three-step approach for evaluating the code snippets at Stack Overflow questions. First, we attempt to find if the Java tagged code actually contains Java code or not. Then we checked if the code snippet contains less than 10 lines of code or not; if so we filtered them out. Finally, matching the criteria according to the Java convention, we evaluated each code snippets separately. We also double-checked our evaluation to ensure consistency. Finally, we had two specialists to analyze our dataset.

Result Analysis

After the evaluation for each Stack Overflow question, we generated a count for good code, bad code and percentage of the effect of good code and bad code respectively. Then we analyzed them against different performance measures of Stack Overflow as question score, Accepted answer etc.

4.1 Question Score vs. Percentage

Our first analysis is correlating between question score of SO site vs average percentage (our manual average readability score). Here, the question score is given by reputed software developers of SO site who thinks the quality of the question is good or it is a good question. And percentage is basically proportion of total count of correct metric over sum of total count of correct and incorrect metric.

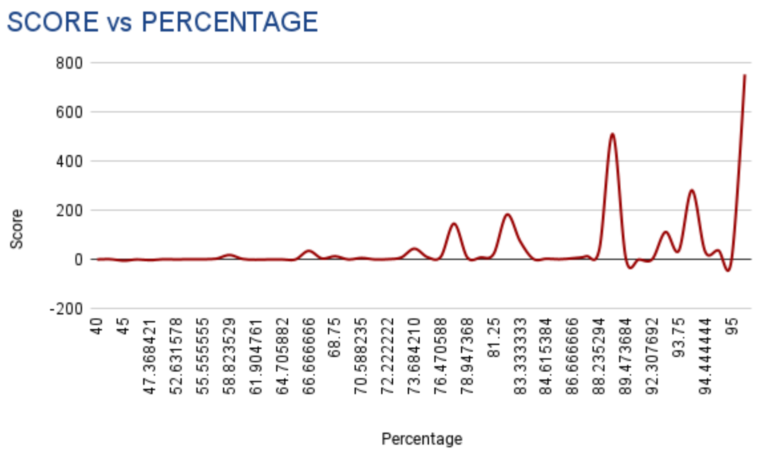

Figure 4.1: Correlation between Question Score and Percentage

From figure 4.1, it is vivid that question score correlates positively with percentage. Moreover, we can see that for percentage less than 73% the score is very low and for percentage greater than 88% the score is very high. With increasing percentage it means to have less incorrect metric and more correct metric. And with higher correct code metric, readability of the question code increases. Therefore, we can infer that high scoring question will be more readable and low scoring question will be less readable.

4.2 Percentage vs Accepted Answer Count

Our second analysis is on correlation between accepted answer of each question of SO site vs average percentage (our manual average readability score). Here, accepted answer means the questioner accepts the answer. Here it is noteworthy that some questions may have answers but may not have any accepted answer. And percentage is basically proportion of total count of correct metric over sum of total count of correct and incorrect metric.

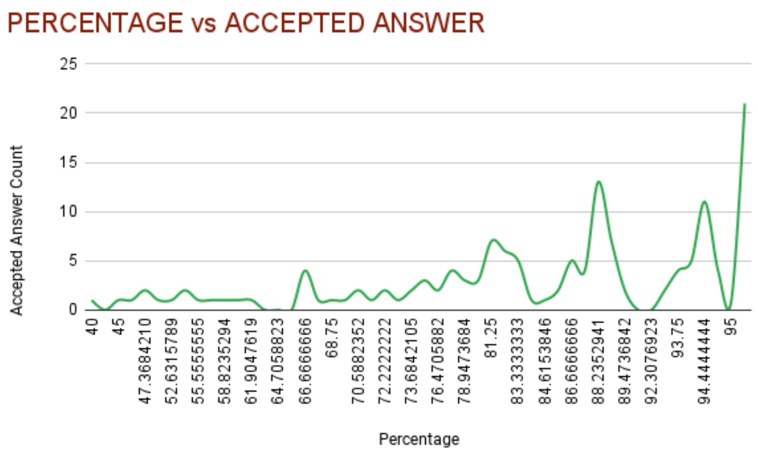

Figure 4.2: Correlation between Question Score and Accepted Answer Count

From figure 4.2, it is vivid that question accepted answer correlates positively with correct code metric percentage. With increasing percentage it means to have more chances to have an accepted answer. Also, we can say with increasing percentage there will be less chances to have an unaccepted answer. By inspecting the figure it is clear that for percentage less than 40 there is no accepted answer. However, higher percentage means presence of correct code metric and with correct code metric, readability of the question code increases. Therefore, we can infer that highly readable question code will likely to have an accepted answer and less readable question will likely to have an unaccepted answer.

4.3 Average Performance on Each Criteria

Our third analysis refers to the average performance of each criteria in our dataset.

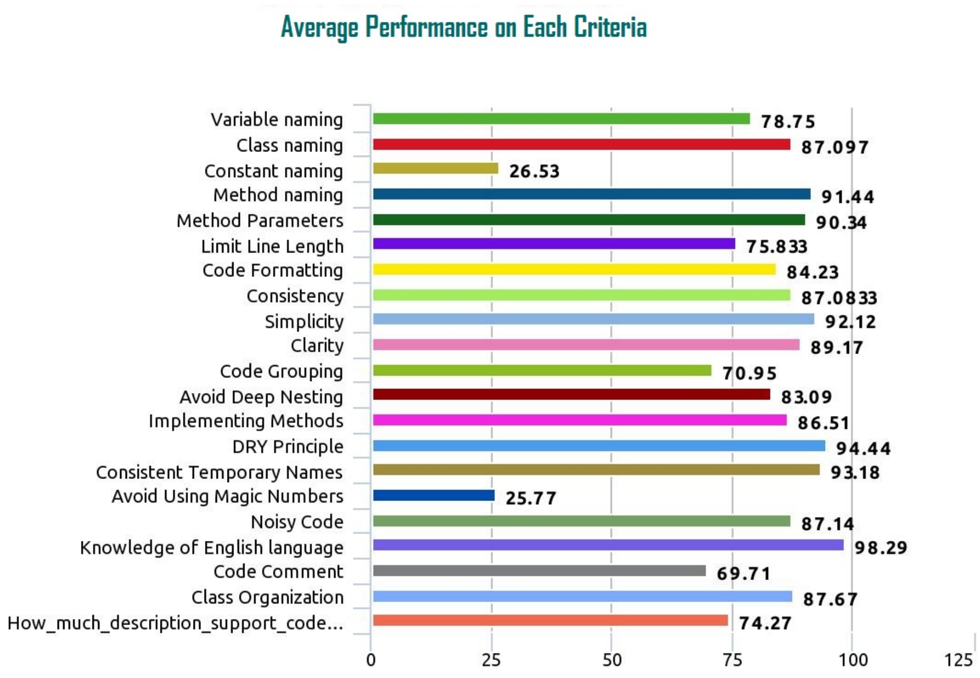

Figure 4.3: Average Performance on Each Criteria

From figure 4.3, it is vivid that except for two all the criteria have a high contribution in our dataset. **Knowledge of English Language** has the highest success rate **98.29%** in Stack Overflow. Similarly, we can identify other criteria where programmers tried their best to maintain the convention. But in case of **Constant Naming (26.53%)** and **Avoid Using Magic Number (25.77%)** we get that programmers often make this mistake to follow the right convention for Java.

4.4 Violation vs Score

Our next analysis is on correlating between violation count of code metric and question score. Here, the question score is given by reputed software developers of SO site who thinks the quality of the question is good or they found it useful. And violation count is total count of code metric that violated readability characteristic.

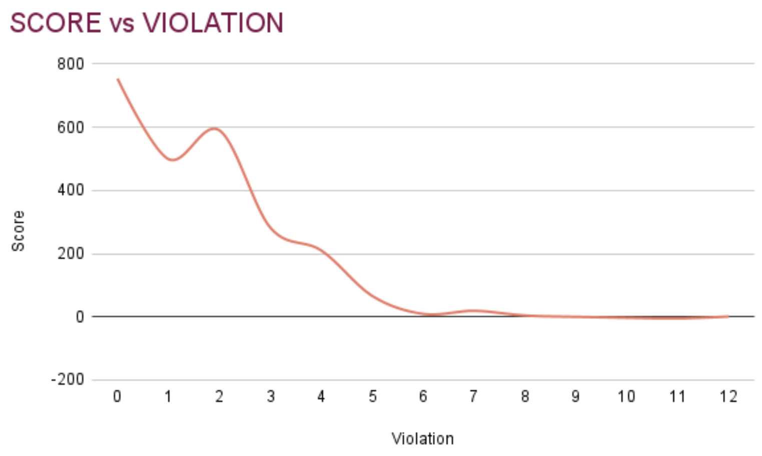

Figure 4.4: Correlation between Violation and Score

From figure 4.4, it is vivid that question score correlates negatively with count of violation. With increasing number of violation, the question score decreases gradually. Also, we can see that when violation count is greater than 5 then question score is close to 0. And when the violation count is equal or less than 2 then question score is higher. Therefore, we can infer that high scoring question will have less violation.

4.5 Violation vs Accepted Answer Count

Our next analysis is on correlating between violation count of code metric and Accepted Answer Count. Here, violation count is total count of code metric that violated readability characteristic. And Accepted Answer Count represents the count of accepted answer for that number of violations.

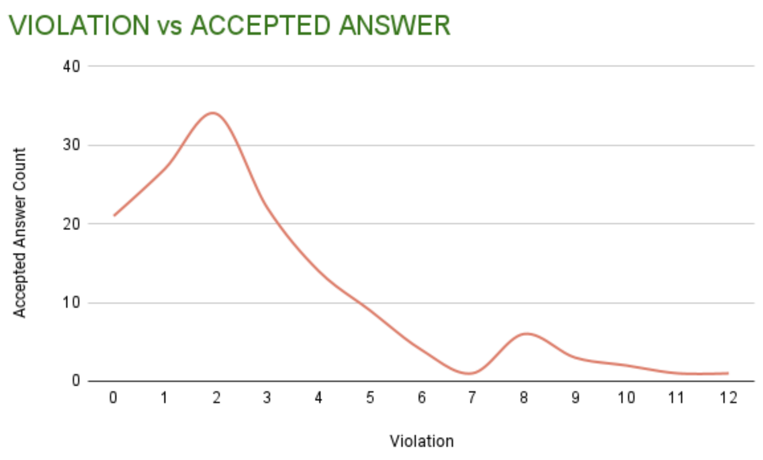

Figure 4.5: Correlation between Violation and Accepted Answer

From figure 4.5, it is vivid that Accepted Answer Count correlates negatively with count of violation. With increasing number of violation, the Accepted Answer Count decreases gradually. Also, we can see that when violation count is greater than 11 then Accepted Answer Count decreases to 0. And when the violation count is 2 or less than accepted answer count is the highest. Therefore, we can infer that for an accepted answer, question code need to have less violation.

Key Findings and Guidelines

We have seen that bad code formatting affects code grouping. Another important finding is that comments can be used for the description purpose. We have seen this technique is used and it's quiet effective for code readability of long code snippets. Another finding is the sequential code description for long code snippets increases the code readability. It's easier to focus on a particular portion and understand step by step. We also found that for serious code readability issues programmers are commenting the criteria to fix to make the code more readable.

While submitting question in Stack Overflow it is recommended that question code to have good code readability. Because, a good code can represent the issue in a short time, and can attract accepted answer of the answering. Also the more the code is readable the less it is error prone and understandable by developer [2, 14, 16]. Our study analyzes further about correlation of code readability with other aspects of SO site.

Threats to Validity

External validity threats are related to a technique's generalizability. Our work consisted of limited number of questions and limited number of annotators. In the vastness of the Java related question in the SO site our limited questions can not generalise all of the questions. Besides, we conducted our study for only Java language. Therefore, if other languages like c++, c# are evaluated by similar manual approach as us then our work may generalise to some extent. However, our work may give important insights to work with other programming languages and we advise readers not to extrapolate our findings.

Internal validity threats are related to experimental mistakes and biases. Threats to internal validity might be if the 3 annotators badly evaluate the dataset with consistency. As a result, when an issue cannot be recreated, we cross-validate our findings and change the evaluation only if the values are inconsistent. So, if the values are bad but consistent then it might be a threat for our work validation.

Conclusion and Future Work

In our work, we introduce a novel dataset Java code readability of Stack Overflow. We manually evaluated 241 Java related questions from Stack Overflow and wanted to establish a correlation of code readability with question scoring and accepted answer count. We evaluated the 21 coding metric in using three indices namely, good, bad, and absent. We investigated how much each criteria have impact on code readability. We find that mostly the criteria namely, (1) Avoid Using Magic Numbers (percentage score **25.77%**), and (2) Constant Naming (percentage score **26.53%**), are readability challenges in asking new question. And the third lowest is code comment with **69.71%** which is also a bit of concern for a questioner as **30.29%** sometimes it gets difficult to understand code without comments or due to obvious or bad comments the code becomes very noisy. But, all other metrics performed well. And among the other metrics the metrics namely, Knowledge of English, Simplicity, Dry Principle, Consistent Temporary Names, Method Parameters,and Method Naming scored above **90%** percentage score. After investigating the correlation between the question score, performance and accepted answer, the overall result shows that code with good readability will have high score and accepted answer. We also correlated the number of violations with question score and accepted answer. And the result is, the more the violations in code metric, the less likely a question can have a good score and an accepted answer. Our findings suggests that questioner should be encouraged to ask question with good code readability.

As we have worked on a smaller dataset so far, we plan to run our experiment in larger dataset of Java code in future and validate our work by using automatic analysis tools supports.

References

[1] M. Duijn, A. Kucera, and A. Bacchelli, "Quality questions need quality code: Classifying code fragments on stack overflow," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 410–413.

[2] C. Treude and M. P. Robillard, "Understanding stack overflow code fragments," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 509–513.

[3] "stackoverflow," https://stackoverflow.com/help/privileges/vote-up, accessed: 2022-04-22.

[4] U. A. Mannan, I. Ahmed, and A. Sarma, "Towards understanding code readability and its impact on design quality," in *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, 2018, pp. 18–21.

[5] S. Fakhoury, D. Roy, A. Hassan, and V. Arnaoudova, "Improving source code readability: Theory and practice," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 2–12.

[6] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on software engineering*, vol. 36, no. 4, pp. 546–558, 2009.

[7] A. Batool, M. Rehman, A. Khan, and A. Azeem, "Impact and comparison of programming constructs on java and c# source code readability," *Int. J. Softw. Eng. Appl*, vol. 9, pp. 79–90, 2015.

[8] J. Johnson, S. Lubo, N. Yedla, J. Aponte, and B. Sharif, "An empirical study assessing source code readability in comprehension," in *2019*

IEEE International Conference on Software Maintenance and Evolution (ICSME). IEEE, 2019, pp. 513–523.

[9] D. Alawad, M. Panta, M. Zibran, and M. R. Islam, "An empirical study of the relationships between code readability and software complexity," *arXiv preprint arXiv:1909.01760*, 2019.

[10] M. U. Tariq, M. B. Bashir, M. Babar, and A. Sohail, "Code readability management of high-level programming languages: a comparative study," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 11, no. 3, pp. 595–602, 2020.

[11] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk, and R. Oliveto, "Improving code readability models with textual features," in *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE, 2016, pp. 1–10.

[12] S. Scalabrino, M. Linares-Vásquez, R. Oliveto, and D. Poshyvanyk, "A comprehensive model for code readability," *Journal of Software: Evolution and Process*, vol. 30, no. 6, p. e1958, 2018.

[13] R. M. dos Santos and M. A. Gerosa, "Impacts of coding practices on readability," in *Proceedings of the 26th Conference on Program Comprehension*, 2018, pp. 277–285.

[14] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 286–28 610.

[15] Y. Tashtoush, Z. Odat, I. M. Alsmadi, and M. Yatim, "Impact of programming features on code readability," 2013.

[16] R. P. Buse and W. R. Weimer, "A metric for software readability," in *Proceedings of the 2008 international symposium on Software testing and analysis*, 2008, pp. 121–130.

[17] "Stack Exchange data explorer," https://data.stackexchange.com/, accessed: 2022-04-22.

[18] “stackoverflow,” https://stackoverflow.com/questions/3280987/spring-separate-datasource-for-read-only-transactions, accessed: 2022-04-22.

[19] “stackoverflow,” https://stackoverflow.com/questions/592496/generating-a-faces-context-manually, accessed: 2022-04-22.

[20] “stackoverflow,” https://stackoverflow.com/questions/3280987/spring-separate-datasource-for-read-only-transactions, accessed: 2022-04-22.

[21] “stackoverflow,” https://stackoverflow.com/questions/3327137/using-for-each-over-an-array-of-objects-integer-array-why-does-forint, accessed: 2022-04-22.

[22] “stackoverflow,” https://stackoverflow.com/questions/3818284/android-surfaceholder-unlockcanvasandpost-does-not-cause-redraw, accessed: 2022-04-22.

[23] “stackoverflow,” https://stackoverflow.com/questions/3818284/android-surfaceholder-unlockcanvasandpost-does-not-cause-redraw, accessed: 2022-04-22.

[24] “Wikipedia,” https://en.wikipedia.org/wiki/Temporary$_variable, accessed : 2022 - 04 - 22.$