Identifying inhibitors and motivators for writing code with high readability and what we can do about it

A study of motivation to write code with high readability

Isac Svensson

June 15, 2022

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 bachelor's degree in software engineering. The thesis is equivalent to 10 weeks of full-time studies.

Contact Information:
Author(s):
Isac Svensson
E-mail: isac.m.svensson@gmail.com

University advisor:
Dr. Nauman Bin Ali
Dept. Computer Science & Engineering

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

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

Abstract

As time passes, more and more code is written, and as companies own a large quantity of code, the importance of readable and easy-to-understand code increases. It is well known that a developer's time is spent primarily on reading and understanding source code. This study aims to answer the questions: do students and professionals in software development agree on the current state-of-the-art of source code readability, what inhibits developers from writing code with high readability, and what activities they perceive to help them write code with high readability. In this context, readability is defined as the ease with which a person can read, understand, and comprehend source code, and the effort it takes to understand the logic, relations, cohesion, and ideas behind the code.

A survey was distributed by email and social media to people working in the field. The respondents were asked to answer questions about readability and three self-evaluating questions about how they work with code. Responses were analyzed and compared to the state-of-the-art in readability of source code.

A literature study was performed to determine the current state-of-the-art of source code readability. The survey result showed considerable support for the findings of the literature review.

The survey results also revealed that the most significant inhibitors for writing code with high readability are connected to lack of time, and the most recognized method of increasing readability is code reviews.

Keywords: Readability, Comprehension, Clean Code

List of Figures

3.1 Visualization of literature search 8

4.1 Occupation of participants 14
4.2 Level of education 14
4.3 Experience in software development 15
4.4 Define what clean code is to you 15
4.5 What programming languages are you able to use proficiently? . . 16
4.6 What programming languages do you use the most? 16
4.7 Length of lines of code 17
4.8 Usage of magic numbers 17
4.9 Usage of arithmetic formula directly in the source code, opposed to encapsulating it within a function. 17
4.10 Writing code in a format that is consistent with the previous code 18
4.11 Usage of comments 18
4.12 No trailing comments 18
4.13 Nesting loops or if-statements 19
4.14 Defining all variables at the top of the method/function, as opposed to defining the variable as close to the first reference as possible . 19
4.15 Usage of Hungarian notation 19
4.16 Names with full words found in dictionaries 20
4.17 Commonly used abbreviations such as “txt”, “msg”, “cnt” or “retVal” 20
4.18 What naming style do you think is easiest to read? 20
4.19 Do you think giving constant variables SCREAMING_SNAKE_CASE names increases readability? 21
4.20 When I have completed a task and commit the code into a repository, I always feel that the code is refactored and written as readable and clean as I possibly can. 21
4.21 What are the reasons for committing code that you are not fully satisfied with? 22
4.22 What are the reasons for committing code that you are not fully satisfied with? 24

List of Tables

3.1 Criteria for literature search 7

Contents

Abstract **i**

1 Introduction **1**
1.1 Background . . . 2
1.1.1 Readability . . . 2
1.1.2 Glossary . . . 2
1.2 Scope . . . 3
1.3 Organisation . . . 4

2 Related Work **5**

3 Method & Design **6**
3.1 Research Questions . . . 6
3.2 Literature review . . . 6
3.2.1 Purpose & goal . . . 6
3.2.2 Search approach . . . 7
3.2.3 Selection criteria . . . 7
3.2.4 Secondary Search . . . 7
3.2.5 Evaluating and selecting the literature . . . 8
3.2.6 Data extraction . . . 8
3.2.7 Search result . . . 8
3.3 Survey . . . 9
3.3.1 Design . . . 9
3.3.2 Targeted Population . . . 10
3.3.3 Sampling . . . 10
3.3.4 Data Analysis . . . 11

4 Results & Analysis **12**
4.1 Literature Review . . . 12
4.1.1 Findings answering RQ1 . . . 12
4.2 Survey . . . 13
4.2.1 Demographics . . . 13
4.2.2 Readability . . . 17
4.2.3 Self-evaluation . . . 21

4.3 Exploitative analysis 24

5 Discussion **25**
5.1 Answers To Research Questions 25
5.1.1 RQ1 - What principles does the science community suggest for increased readability in source code, that are applicable when writing code? 25
5.1.2 RQ2 - To what extent do IT professionals agree with the suggested findings? 25
5.1.3 RQ3. Do IT-professionals experience that they always commit readable code? If not, what are the reasons for committing code that they do not consider as readable? 27
5.1.4 RQ4. What countermeasures do IT professionals suggest to help them increase readability in code? 28

6 Threats to validity **29**
6.1 Not a representative sample 29
6.2 Open invitations on social media 29
6.3 Not defining and describing concepts 29
6.4 Presumption of a shared definition of readability 30
6.5 Human error 30
6.6 Internal bias 30
6.7 Sabotage & Hypothesis Guessing 30

7 Conclusions & Future Work **31**

A Survey questions **36**

Chapter 1

Introduction

As time passes, more and more code will be generated, and as companies own a large quantity of code, the importance of readable and easy-to-understand code increases. If you do not take care of your codebase from the start, you will end up with messy code and unnecessary technical debt. There is no consensus regarding what constitutes clean code. However, a common theme in the definitions is the emphasis that clean code is easy to read, understand, and modify [13, 17, 25, 26].

In [25], Martin writes that "Names in software are 90 percent of what make software readable". He writes about naming conventions, using actual words describing what the identifier does or holds, skipping magic numbers and placing them in variables to explain what they are, and being consistent throughout the project. He also writes about vertical and horizontal formatting and density and encapsulation as key components of writing readable and comprehensible code. These statements can be backed up by research from multiple researchers [5, 10, 21, 33].

Grady Booch, author of Object-Oriented Analysis and Design with Applications describes clean code as follows:
"Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control" (as quoted by Martin [25]).

Do IT professionals agree with the findings of the state-of-art research? Do they think the code they commit to a project is always readable? If not, why? We will examine this to determine what role readability and comprehensibility have in the IT professional's everyday life when writing code. The literature review will extract suggested principles to increase code's readability or comprehensibility. We will use the extracted principles in a survey to determine whether IT professionals agree with the suggested principles or not.

1.1 Background

1.1.1 Readability

The word readability has been defined in multiple ways over the years, and when referring to it, it is most often used for describing text. Some definitions are broad and general, while others are more limiting.

Merriam-Webster defines readable as “able to be read easily” [3] and Wiktionary defines it as “The property of being capable of being read; legibility” or “The property of being easy or engaging to read” [2].

Dale and Chall described “..readability is the sum total (including the interactions) of all those elements within a given piece of printed material that affects the success that a group of readers has with it. The success is the extent to which they understand it, read it at an optimum speed, and find it interesting” [14].

In [16], DuBay writes that readability is “..what makes some texts easier to read than others.” and also “..readability is the ease of reading created by choice of content, style, design, and organization that fit the prior knowledge, reading skill, interest, and motivation of the audience”.

Klare defines readability as “..the ease of understanding or comprehension due to the style of writing” [19]. While Guzzetti writes with slightly different words in [18], “readability is the ease of reading comprehension due to the style of the writer”.

When referring to readability in this study, we build upon the definition given in ISO/IEC/IEEE International Standard – Systems and software engineering – Vocabulary," in ISO/IEC/IEEE 24765:2017 [1], “ease with which a system’s source code can be read and understood, especially at the detailed, statement level”, and consider include both understandability and comprehension a part of the term.

1.1.2 Glossary

DRY – "Don’t repeat yourself", This means that if you write the same code more than once, you should consider abstracting the code block into a class or function [25].

KISS – "Keep it simple stupid", or in other words, less is more. Always choose the simplest possible solution.

SoC – "Separation of Concerns", Probably coined by Edsger W. Dijkstra in [15], “...‘focusing one’s attention upon some aspect‘: it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect’s

point of view, the other is irrelevant...‘. Meaning that most components of a program does not need to know anything about the rest of the program.

SOLID – SOLID is an acronym for five design principles with the purpose to make software designs more flexible, understandable, and maintainable.[**?**] The acronym is for:

The **S**ingle-responsibility principle,

the **O**pen–closed principle,

the **L**iskov substitution principle,

the **I**nterface segregation principle, and

the **D**ependency inversion principle.

POC – "Proof of concept", a demonstration of the practical possibility of an idea.

1.2 Scope

The purpose of this study is to gain an understanding of readability in source code, both in theory and in practice, why code with low readability is committed to projects and what we can do to prevent it. Moreover, what practices and guidelines are suggested by the current research, to increase readability in source code. What practices and guidelines to avoid is also included in the scope, i.e. practices that are suggested to decrease readability of source code. The study will only include practices and guidelines that are applicable when writing code (i.e., not measures for analysis of already completed code) on a statement level.

Principles for code formatting such as statement grouping, indent size and white space rules are not included in the scope of this study.

Furthermore, the study aims to determine to what extent people who frequently write, maintain and/or review others code (referenced as IT professionals after this) agree to the suggested practices. The scope includes people working in the area of software development, computer science students and practitioners engaging in software development. People without experience in the area of software development and people without frequent interactions with source code are not included in the scope.

The study will examine how the participants evaluate themselves regarding commitment of readable code, what the reasons are to commit code with low readability and what countermeasures the participators suggests to help them increase readability in the code they produce.

1.3 Organisation

The remainder of the theses is organized as follows:

Chapter 2, Related Work – a brief review of related work in the areas of readability in source code, how to measure readability in source code, and motivators for refactoring code are presented.

Chapter 3, Method and Design – presents the research questions, chosen methods for the study, and the design and processes for the chosen methods.

Chapter 4, Results & Analysis – we go through, analyze, and present the results of both the literature review and survey.

Chapter 5, Discussion – A thorough discussion of the result and answers to the research questions are presented in Chapter 5.

Chapter 6, Threats to validity – Some possible threats to validity are examined in Chapter 6. We are looking at risks in the sampling methods and possible problems in the survey.

Chapter 7, Conclusions and Future Work – In the last chapter, we discuss some possible ways to extend the thesis and confirm the conclusions further. We also conclude and summarize the thesis.

Appendix A, Survey Questions – An appendix is added to the thesis with all the questions used in the questionnaire-based survey.

Chapter 2

Related Work

In two studies by Buse and Wiemer, they show "how to produce a metric for software readability from the judgments of human annotators" [9, 10]. They let 120 annotators score 100 code snippets and, from that, extracted features that affect readability. They employed machine-learning techniques to develop a model for predicting readability based on the features and trained with the annotations from the human reviewers.

Posnett et al. built their study [28] based on the work of Buse et al. [10]. However, Posnett et al. adopted three other metrics (Halstead's volume metric, lines of code, and entropy of code tokens or characters) and improved the performance of readability prediction. They proposed a formula for readability prediction in [28] that was verified by using 120 people and 100 code snippets as performed by Buse et al. Posnett et al. estimated that their formula had an accuracy of more than 80% better result than the previous result from the formula by Buse and Weimer [10].

Lee et al. [24], in a study, applied Posnett et al.s' formula [28] to determine the impact of violations in coding conventions. They concluded that some particular coding violations strongly impact readability, such as trailing comments, indentation, and Javadoc convention.

Ribeiro and Travassos did a systematic literature review [29] where they identified source code attributes impacting the source code readability and comprehension and then performed an empiric study to support the attributes that had contradictory results in the literature review. With the empirical study, they suggested explanations for the contradiction in the previous results.

Wang studied the motivations for software engineers to refactor code in [36]. He presented six intrinsic motivators and six extrinsic motivators for software engineers to refactor code. The motivators extracted from the study span from unconscious habits and self esteem on the intrinsic side to recognition from others and threats of punishment on the extrinsic side.

Chapter 3

Method & Design

3.1 Research Questions

The main objective of this study is to gain an understanding of the reasons for committing readable (or not readable) code when working on a project. To achieve this goal, we have defined the following research questions:

RQ1. What principles does the research community suggest to increase readability in source code, that are applicable when writing code? (I.e. not metrics for post-analysis of source code)

RQ2. To what extent do IT-professionals agree with the suggested findings?

RQ3. Do IT professionals experience that they always commit readable code? If not, what are the reasons for committing code that they do not consider readable?

RQ4. What countermeasures do IT professionals suggest to help them increase readability in code?

3.2 Literature review

The literature review has primarily identified systematic literature reviews on source code readability and related constructs like comprehension and understandability.

3.2.1 Purpose & goal

The purpose of the literature review was to identify and extract principles or methods used to increase readability or comprehension in source code. The goal was to extract a list of principles that either increase or decrease readability or comprehension. The list of gathered principles is used to answer RQ1 and is also used in the empirical study in the questionnaire-based survey to answer RQ2, RQ3, and RQ4.

3.2.2 Search approach

The search was performed in Scopus with keywords and conditions below:
("code" OR "program") AND
("comprehensibility" OR "readability" OR "understandability") AND
("literature review" OR "systematic mapping")

The first condition ("code" OR "program") was chosen to increase the likelihood of getting relevant matches by excluding studies that do not explicitly mention code or program.

The second condition serves to choose all studies in the three closely related topics comprehension, readability, and understandability. This with the expectation that even though readability might not be mentioned in the title or abstract, it might contain valuable information.

The condition "literature review" OR "systematic mapping" selects only the studies containing literature reviews. The initial idea was to search with the keyword "systematic literature review", but that is limiting the scope severely, and thus, the first word was excluded.

3.2.3 Selection criteria

Table 3.1 displays the criteria for assessing the relevance of the search results.

Include	**Exclude**
Peer-reviewed journals Conference Workshop papers	Any not peer-reviewed paper
Literature review, Systematic Literature Review, Systematic Mapping	Any paper without methods specified under include section
Computer Science (Software Engineering, development, programming, etc.)	All papers without direct connection to the field of Software Engineering
English or Swedish.	Any other language
≥ 2017	< 2017

Table 3.1: Criteria for literature search

3.2.4 Secondary Search

The selection criteria were also applied to the citations to the literature found in the primary search. The secondary search was performed to increase the likelihood of capturing any recent literature relevant to the study that was not a result of the primary search.

3.2.5 Evaluating and selecting the literature

Resulting from the application of the criteria for inclusion and exclusion, further modification of the literature list was achieved through careful evaluation and selection of the gathered literature.

3.2.6 Data extraction

From the literature collected two types of data were extracted and synthesized:

- Principles/practices/guidelines for increased readability, comprehensibility, and/or understandability of source code.
- Principles/practices that decrease readability, comprehensibility, and/or understandability of source code.

Only principles that are applicable when writing code are considered valuable data (i.e., not measures for analysis of already completed code).

To get the best possible result, each paper referenced and used in the literature reviews was processed and where possible relevant data were extracted.

3.2.7 Search result

The primary search resulted gave 32 papers. After selection from the defined search criteria, three papers remained [20, 29, 30]. The secondary search generated a result of 26 new studies. Out of these 26 studies, two passed the criteria for inclusion [4, 27]. Leaving us with a total of five papers.

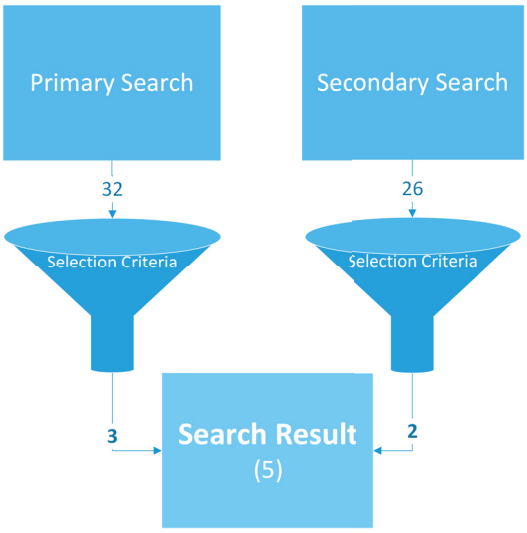

Figure 3.1: Visualization of literature search

3.3 Survey

The empirical part of the study is a questionnaire-based survey. The method for administration is self-completion [31].

All participants in the survey are anonymous. The survey gathered demographics on all participants regarding the level of education, experience in the field, and current occupation.

The next part of the survey presented the participants with different principles that state-of-the-art research suggests to increase or decrease readability in source code. The participants answered to what extent the principle has an impact on source code readability.

Presenting a definition of readability to the respondents would influence their answers regarding what they consider is code readability. Therefore, we decided to collect their definition and perception regarding readability without presenting our definition for it.

3.3.1 Design

The survey consists of three parts

The demographics part – Gathering information about the participants.

Rating of principles – The second part lets the participant rate different principles with what they think the impact on readability is. This is followed by a few questions about different naming conventions *RQ2*

Self-evaluation – The last part asks the participants if they always commit code that they are fully satisfied with (regarding clean code and readability). This is followed by questions about why they commit code they are not satisfied with and what would help or motivate them to commit "cleaner" and more readable code. *RQ3 and RQ4*

Types of Questions

Demographic questions – Ten demographic questions are placed at the beginning of the questionnaire to gather information about the participants. The questions are about occupation, role, experience in the field, education, and a question about reading disorders to identify outliers.

Multiple choice questions/Multiple answer questions – There are eight questions qualified as multiple choice or answer questions. The questions have multiple predefined answers to choose from and if none of the predefined answers fit there is always another option or the possibility to leave an open

answer. In multiple-choice questions, you are only allowed to leave one answer, but in multiple-answer questions, you are allowed to leave multiple answers.

Open questions – In the questionnaire there are two completely open questions. These are placed in the demographic section before the rest of the questionnaire to avoid biasing the participants.

Closed questions – Three questions qualify as closed questions, asking yes- or no questions.

Likert scale questions Twelve questions qualify as Likert scale questions. They are set to a seven-degree scale, where one represents strong negative impact or strongly disagree and seven represent strong positive impact or strongly agree. Some of the questions are semantically negated to avoid bias caused by positive psychology [31].

3.3.2 Targeted Population

The targeted population for the survey is all IT professionals, students, and people that in one way or another handle code. In the targeted population, the subjects have occupations as developer, tester, scientist, teacher or educator, or manager or leader of developers.

3.3.3 Sampling

The methods used for sampling are convenience sampling and snowball sampling.

Convenience sampling is performed by "..choosing the nearest and most convenient persons to act as respondents" [31]. In this study, this was achieved by sending personal invitations by email, inviting friends and co-students, and posting public invitations to participate in the survey in relevant forums on Facebook and LinkedIn.

Robson and McCartan describe Snowball sampling as "..the researcher identifies one or more individuals from the population of interest. After they have been interviewed, they are used as informants to identify other members of the population, who are themselves used as informants, and so on" [31]. In this study, all participants were asked to invite people in the targeted population. The personal email invitations were sent to managers for development teams in different organizations. In addition to the invitation, they were asked to identify and invite other members of the population in their organization to invite.

3.3.4 Data Analysis

The chosen approach for data analysis in this study is exploratory analysis. In contrast to traditional confirmatory analysis, exploratory analysis explores the data trying to find out what the data can tell you [31].

Analysis has been performed by filtering the data-set on the demographic questions and comparing the results between different groups and the result gathered in the literature study.

Since the chosen sampling approach is convenience sampling and no conclusions about the population can be made, this paper only analyzes and discuss descriptive statistics.

Quantitative Data

The quantitative data has been analyzed by selecting groups from the demographics questions and comparing the data from the given answers in the groups to both other groups and the result from the literature review.

The data was plotted to pie charts, Likert-scale charts, and histograms for easy comparison. Medians and means were also compared.

Qualitative Data

For the qualitative data thematic coding [31] was chosen as the approach for extracting data. All answers were read and analyzed by breaking the answer down into smaller statements and categorizing them.

Chapter 4

Results & Analysis

4.1 Literature Review

The literature review served to answer **RQ1** and to give a foundation to build the questionnaire survey upon. The result was gathered by collecting principles or methods to increase readability that are applicable when writing the code. The findings are presented in the following section.

4.1.1 Findings answering RQ1

Short lines – Buse and Weimers' findings in [9, 10] support the conventional advice to write short lines of code, even if it means breaking up a statement across multiple lines.
Constants instead of magic numbers & upper case rather than snake case – Conforming constant naming to a specific format increases readability [17, 23]. For example, using the lowercase constant name max_value = 100 is better than hard coding 100 directly in the source code. But the studies support that using uppercase style and naming the constant MAX_VALUE = 100 grants the best possible readability.
Meaningful names – The claim that the usage of meaningful names that describe the functionality and/or purpose of the function, object, or variable is widely supported both by the clean code community [25] and in research [5, 21, 34].
Full word or abbreviation identifiers rather than single letter – In [22] findings show that identifiers composed of full words or abbreviations lead to better comprehension than single letter identifiers. Findings in [11, 21] also support that the use of natural language and recognized abbreviations in identifier names increases readability and comprehension.
Consistent style – Using the same style as the previous code increases readability more than using another style than the previous code, even if the style used in the new section is suggested to be more readable [25, 34].
Comments – In [12, 34, 35, 37], researchers all agree that comments increase readability and comprehension of source code. However, the findings of [24] sug-

gest comments decrease readability. The experiments results suggest that violations of the Javadoc comments guide increase readability rather than decreasing it.

Trailing comments – Trailing comments attached on the same line as code are indicated to not be a good practice in [24]. The experiments showed that placing comments on separate lines from the code increased readability.

Recursive functions, nested loops, and arithmetic formulas – Recursive functions, nested loops, and arithmetic formulas were found to be of a negative impact on the general readability attribute [34].

Distance between definition/declaration and first reference of a variable – Findings in [32] suggest that smaller distances between the definition of a variable and the first reference of it increase readability.

Naming style – The discussion about naming styles is divided, but the majority suggests that camel case style produces the best readability [7, 8, 12]. In [38] findings suggest that the choice between underscore style and camel case style does not impact the subjects effort and program comprehension. And a study on naming styles using eye-tracking [6] suggests that underscore style benefits the subjects' in respect of time spent.

Hungarian notation – Buse and Weimer [9] suggest that the net gain for using encoding of types into identifiers is near zero. They suggest that it might even decrease readability. In another study [10] they used length of an identifier as a predictor of readability, and the result showed that long identifiers are connected to lower readability. They did not include any features to detect encoded type information but found that longer names without conveying extra information may have a negative impact on readability. Butler et al. [11] found that no impact on readability was observed when using type-encoded names.

4.2 Survey

The survey was open for answers for a week, between April 25th and May 2nd, 2022 and received 200 responses. Two of the respondents answered the question about "Role and previous experience" that they did not either write, debug, read or review code frequently. These respondents' answers are removed, resulting in a total of 198 answers, which was much higher than expected.

4.2.1 Demographics

Occupation

The bigger part of the participants were developers. And the biggest group after that was students (see figure 4.1).

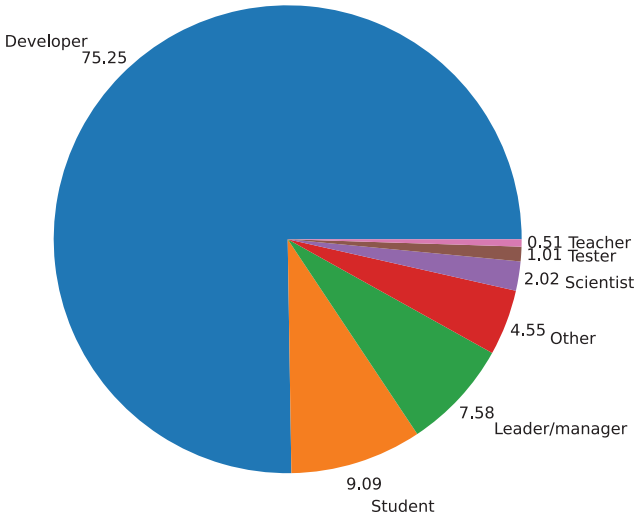

Figure 4.1: Occupation of participants

Education

The collected data shows that over 90% of the participants have completed some kind of higher education. Two-thirds of the participants have received an academic degree and 42.5% have received at least a master's degree (see fig 4.2).

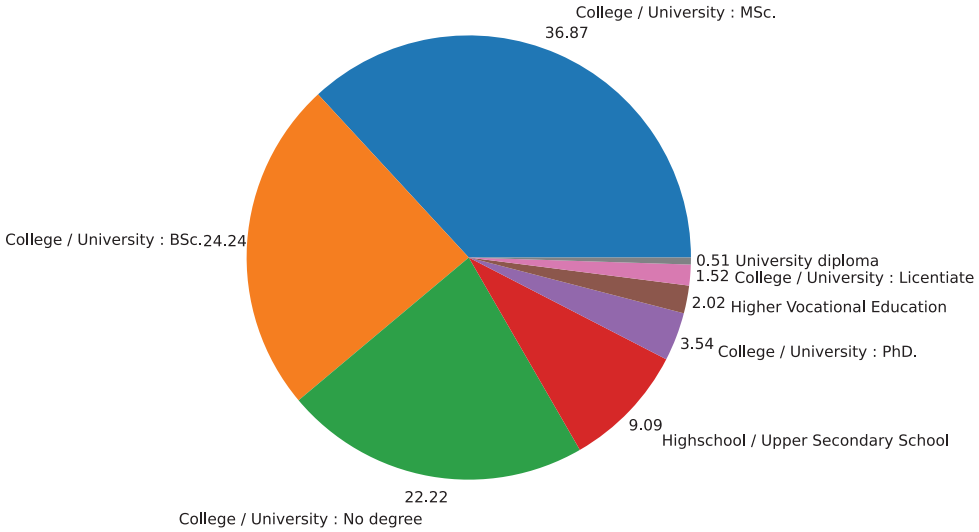

Figure 4.2: Level of education

Experience in software development

The experience of participants differs greatly. The minimum registered experience is less than a year and the maximum is 40 years. The mean experience is 14.83 years and the median is 12.5 years (see fig 4.3)

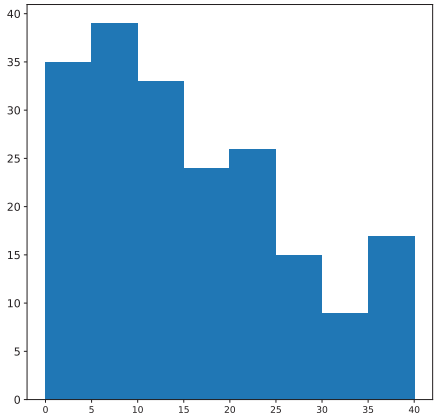

Figure 4.3: Experience in software development

Previous knowledge of clean code

When asked about clean code, 87.37% of the participants answered that they are aware of the term. When prompted to give their definition of the term, the principles brought up ranged from easy to read to the usage of Hungarian notation. The five top categories for describing clean code were easy to read (readability), easy to understand (comprehensibility), clear and meaningful names, easy to maintain (maintainability), and short functions and classes. In figure 4.4 all categories mentioned more than once are displayed.

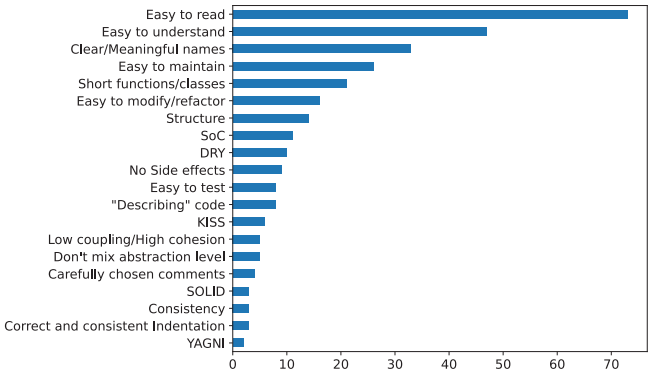

Figure 4.4: Define what clean code is to you

Programming languages

The top of the answers for what programming languages the participants use the most and what languages they can work proficiently with are almost identical. The only answer that differs among the 15 highest ranked languages is that 20 people answered that they know Visual Basic, but since only two people said that it is among their most used languages, it did not make the top 15 languages worked with. Rust is the language that differs in the top of what people use the most. Eight out of nine people that specified Rust as a language they can work proficiently with also said that Rust is among their most used languages.

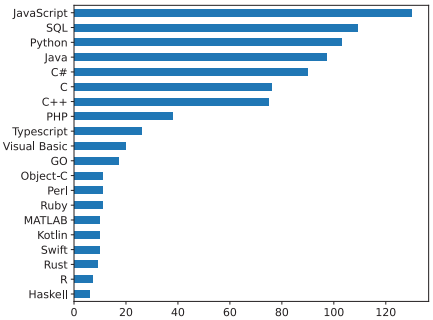

Figure 4.5: What programming languages are you able to use proficiently?

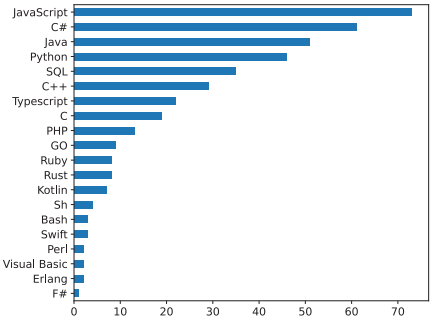

Figure 4.6: What programming languages do you use the most?

Reading disorder

Twelve participants answered that they have some kind of reading disorder, and five more answered that they do not know if they do.

4.2.2 Readability

Short lines

In the question about whether keeping all lines of code short or occasionally writing long lines (fig 4.7), 70% of the participants answered that, to some agree, long lines have a negative impact on readability.

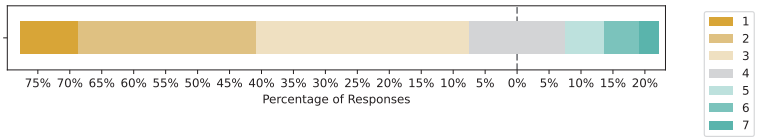

Figure 4.7: Length of lines of code

Magic numbers

In the question about magic numbers, or "Usage of numbers directly in the source code, opposed to first encapsulating it within a constant or a variable", as the question is stated in the survey, 67.7% of the participants said that it has a negative impact on readability. Looking at the other end of the scale, 9.6% state that they think it has a positive impact, leaving the last 22.7% neutral. Fig 4.8

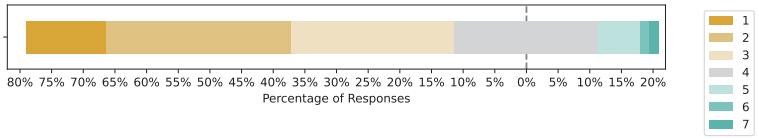

Figure 4.8: Usage of magic numbers

Arithmetic formulas directly in the source code

Almost the same numbers are reported for using arithmetic formulas directly in the source code, as opposed to encapsulating them within a function. 63.4% answered that it has a negative impact as opposed to 9.6% who think that it has a positive impact. Fig. 4.9

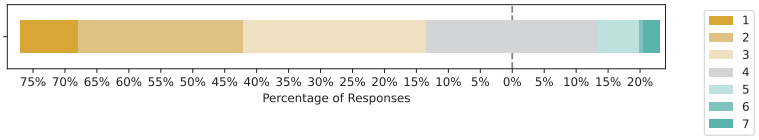

Figure 4.9: Usage of arithmetic formula directly in the source code, opposed to encapsulating it within a function.

Consistent code

"Writing code in a format that is consistent with the previous code" was the question where the participants had the highest agreement, 91.4% answered that this had a positive impact on readability. Fig 4.10

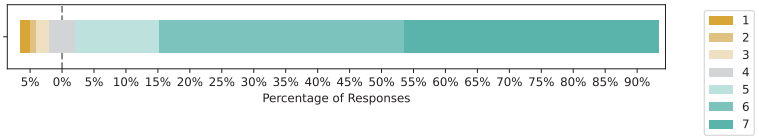

Figure 4.10: Writing code in a format that is consistent with the previous code

Comments

When asking about usage of comments, 53% thinks that comments increase readability. Many participants added reservations about this question, specifying that comments should be selected carefully and not be used when you can use identifiers and names to convey the same information. Fig 4.11

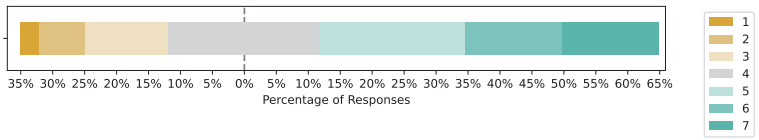

Figure 4.11: Usage of comments

On the other question about comments, whether comments should be placed on their own separate rows or be allowed to be "trailing" a line of code, the majority chose to stay neutral (45.7%). 35% of the participants answered that they think writing comments on their own rows has a positive impact on readability and the remaining 19.3% think the opposite. Fig. 4.12

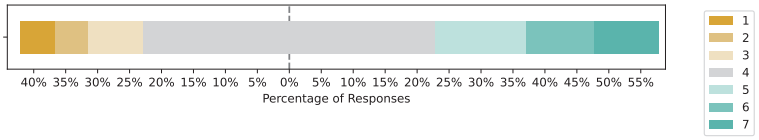

Figure 4.12: No trailing comments

Nested loops and if-statements

The question about nested loops and if-statements had 66.2% negative responses and only 7.1% who thought that it has a positive impact on readability. Fig. 4.13

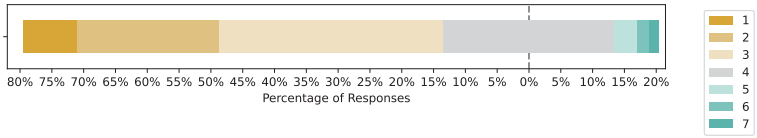

Figure 4.13: Nesting loops or if-statements

Distance between definition and first reference

54% of all participants think that "Defining all variables at the top of the method-/function, as opposed to defining the variable as close to the first reference as possible" has a negative impact on readability, while 25.6% say the opposite. Fig. 4.14

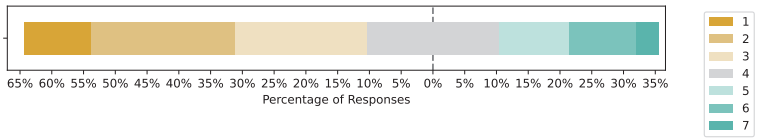

Figure 4.14: Defining all variables at the top of the method/function, as opposed to defining the variable as close to the first reference as possible

Hungarian notation

16.67% think that "Hungarian notation" has a positive impact on readability, while 60.6% think the opposite. Fig. 4.15

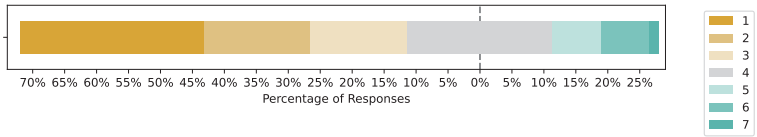

Figure 4.15: Usage of Hungarian notation

Full words found in dictionaries

Most participants (73.23%) think that using names with full words that can be found in a dictionary has a positive impact on readability. Only 7.6% think that this has a negative impact. Fig. 4.16

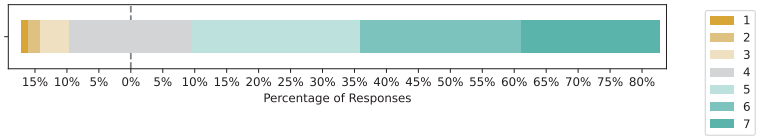

Figure 4.16: Names with full words found in dictionaries

Commonly used abbreviations

The question about commonly used abbreviations such as "txt", "msg" or "cnt" got the most even result. 40.4% answered that they think commonly used abbreviations have a positive impact on readability, while 37.9% answered that they think it has a negative impact. Fig. 4.17

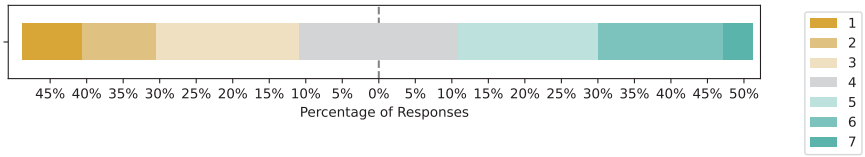

Figure 4.17: Commonly used abbreviations such as "txt", "msg", "cnt" or "retVal"

Naming style

When asked about what naming style is easiest to read the majority (54.4%) answered that it "does not matter, as long as we are consistent". Other than that the by far highest rated style was camel case with support from 24.75% of the participants. Fig. 4.18

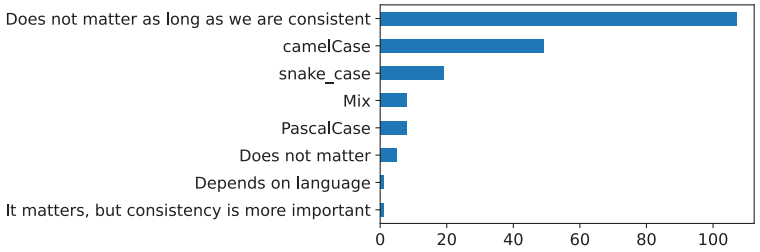

Figure 4.18: What naming style do you think is easiest to read?

Naming style on constants

The last question on readability was, "Do you think giving constant variables SCREAMING_SNAKE_CASE names increases readability?". To this question the answers were split between 61.11% agreeing, and 38.89% disagreeing. Fig. 4.19

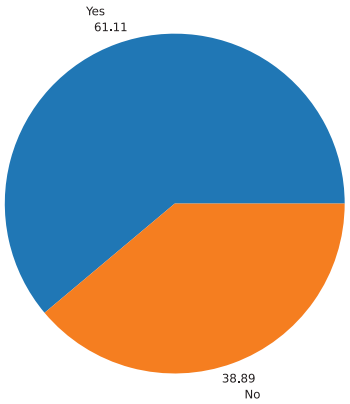

Figure 4.19: Do you think giving constant variables SCREAMING_SNAKE_CASE names increases readability?

4.2.3 Self-evaluation

When asked about the statement "When I have completed a task and commit the code into a repository, I always feel that the code is refactored and written as readable and clean as I possibly can.", 67.17% agree at some level, 13.13% agree fully (see fig. 4.20). 25.25% chose a value on the negative side of the scale. The big difference from the result on the reasons for committing code they are not satisfied with (fig. 4.21) for all participants, is that in the full result 14.07% say that it is not worth the effort, while in the negative span, 22% says the same thing.

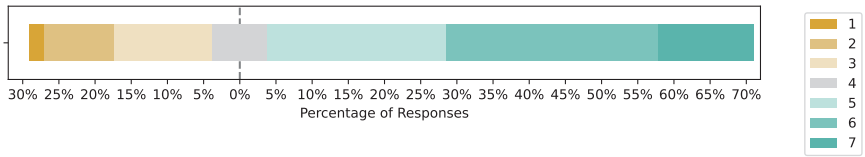

Figure 4.20: When I have completed a task and commit the code into a repository, I always feel that the code is refactored and written as readable and clean as I possibly can.

Reasons for committing "bad" code

The reasons why the participants commit "bad" code are many, but the by far biggest reason is working under an unrealistic schedule (fig. 4.21). The two following reasons are in the same category, too small budget and lack of planning. The fourth most common reason is that it simply is not worth the effort to write code that you are fully satisfied with. Many participants write about being

satisfied with "good enough". Some participants say that they often write code with a short lifetime (prototypes and PoCs) and that functionality and time to finish product are the priority. Others say that they are working with big code bases of legacy code and that cleaning the code is an iterative process.

To this question, the answers in the list below were predefined and given to the participants.

- Lack of planning
- Lack of interest
- Lack of training or knowledge
- Unrealistic schedule
- Too small budget
- Code readability is not considered as a concern in my organisation/team
- Writing clean and readable code is not encouraged in my team/organisation

The participants were also able to write their own answers in free text. These answers were coded using thematic coding and placed in one or more of the predefined alternatives. Based on the answers four new categories where created and the answers were categorized into these too.

- Not worth the effort
- Working with legacy code
- I, or someone else will clean up later
- POC/Prototype/Experiment

Single answers not fitting in any of the other categories were put in the "other" category.

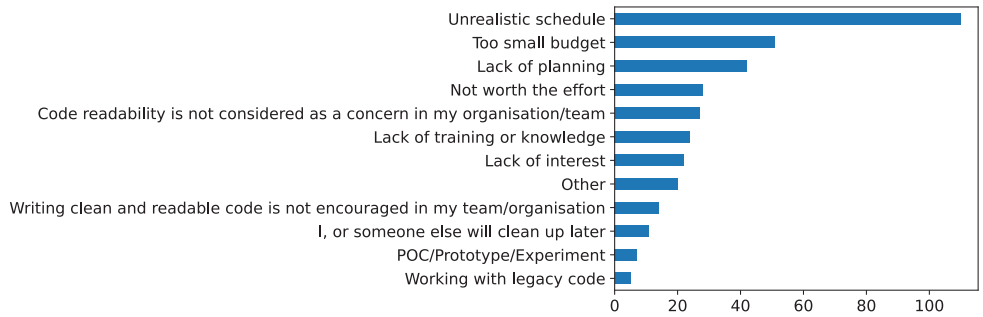

Figure 4.21: What are the reasons for committing code that you are not fully satisfied with?

Activities for writing more readable code

The open answers to the question about activities that help write readable code were coded, as previous answers into primarily predefined categories (see list below) or into new categories based on the answers.

- Code review
- Pair programming
- Education and seminars
- More time
- Better tools

The new categories created were:

Experience Some participants wrote that experience or learning by doing is a good way to go. One participant expressed it "Maintaining your own code ... Is the only way to for ppl to learn to care!"

Clearer standards Some participants said that agreeing to work with a predefined standard across team/company is important

Self-review Another suggested activity was self-review. I.e. to read through and review your own code an extra time before committing.

Some other suggestions were to write an alternative implementation and take the best of them both or have requirements and static code analysis in CI.

The most frequent answer, given by 82.8% (fig. 4.22) of the participants, was "code reviews". This seems to be by far the activity that is perceived as the most important activity to increase readability in the code but also code quality overall. Other related activities discussed in open answers are pair programming and mob programming.

The second most given answer correlates to the answers to the previous question, "more time". Given more time, a more realistic schedule could be made and planning could have been given some extra time.

Better tools and software was brought up by many participants. Mainly auto formatters, such as "black" or "prettier", and linters were given as examples.

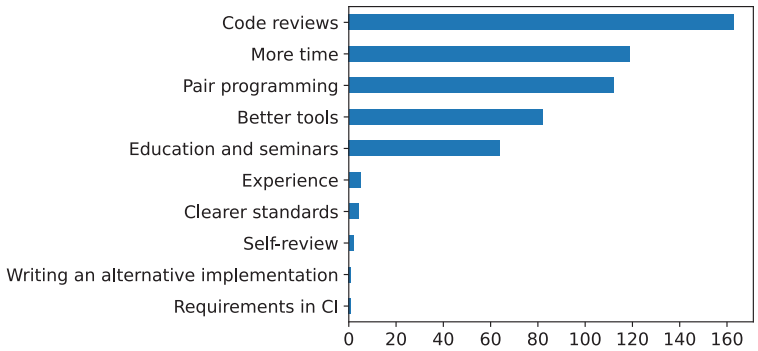

Figure 4.22: What are the reasons for committing code that you are not fully satisfied with?

4.3 Exploitative analysis

The exploitative analysis found no significant difference in opinions across the answers when comparing answers by participants with different occupations except for the difference between students and developers in the question about Hungarian notation (this is described and discussed further in Subsection 5.1.2).

Comparing the answers between participants grouped by years of experience (five-year spans) there was no obvious difference in the answers.

An analyze on based on what programming language the participant is used to work with showed a minor difference in the opinion about the impact of defining all variables at the top of the function instead of close to the first reference. Participants programming in C and JavaScript had a bigger portion of positive answers than the sample.

Chapter 5

Discussion

5.1 Answers To Research Questions

5.1.1 RQ1 - What principles does the science community suggest for increased readability in source code, that are applicable when writing code?

See the result under Subsection 4.1.1.

5.1.2 RQ2 - To what extent do IT professionals agree with the suggested findings?

Short lines – 70% of the participants answered that long lines have a negative impact on readability. This correlates to the result of the literature review.

Constants instead of magic numbers & upper case rather than snake case – 67.7% of the answers given for the question about usage of magic numbers were stating that it has a negative impact on readability while only 9.6% states the opposite. The answers about if screaming snake case for constants increases readability ar not as clear. The positive side of the question is a majority, but the 38.89% is still a considerably high percentage of people who do not think it increases readability. The result of the questions correlates with the result from the literature study. Magic numbers are described as having a negative impact on readability and giving constants names with screaming snake case have a positive impact on readability.

Meaningful names – A question about this in the survey was omitted on purpose. The topic was left out because of the inability to write a short, concise, and clear enough question to describe the practice in the questionnaire. But, looking at how the participants described their definition of clean code (fig. 4.4), this shows that meaningful names was the third most brought up category. This *could* implicate that there is a support for this thesis.

Full word or abbreviation identifiers rather than single letter – The result for the question about using full words found in dictionaries confirm that the participants of this questionnaire agree with the statement. When asked about using common abbreviations in names the opinions are split and no conclusion can be made and neither confirms or dismiss the findings from the literature review.

Consistent style – The question about writing code in a format that is consistent with previous code got the biggest support of all questions with over 90% thinking that it has a positive impact on readability. This correlates with the findings of the literature study.

Comments – A clear majority of the participants think that comments increase readability (fig 4.11). This correlates with previous findings in [12, 34, 35, 37]. In the answers to the question about best practices and clean code, some participants wrote that comments only should be used when necessary and that comments should be used only when the information can not be expressed in code with for example meaningful names. This falls in line with the findings in [24], which suggests that strict guidelines for commenting all components might decrease readability.

Trailing comments – 45.7% of the participants chose to be neutral in this question. Out of the remaining participants, the bigger part (35%) answered that they think that not allowing trailing comments increases readability (fig. 4.11). The findings indicate possible support for the findings in the literature review.

Nested loops – The question about nested loops got answers summing up to 66.2% thinking that nesting of loops and if-statements have a negative impact on readability. It should be noted that comments were made outside the survey that this is not always the case. The opinion expressed was that sometimes multiple-level nesting is the best option. The result of the question supports the findings in the literature review.

Arithmetic formulas – 63.4% of the participants answered that they think the usage of arithmetic formulas directly in code has a negative impact on readability (fig. 4.9). This supports the findings in the literature review.

Distance between definition/declaration and first reference of a variable – 54% of the participants answered that this has a negative effect on readability to define all variables at the top of a method, as opposed to defining it as close to the first use of it as possible. This supports the findings in the literature review. Something more surprising is that 25.6% of the participants think it has a positive impact. One possible reason for this is that people working with languages that have a best practice of defining variables at

the top of a function are accustomed to this, expect this and thus think it is more readable when written this way. A group that has higher answers in this group than the sample is people working with C and people working with JavaScript and TypeScript.

Naming style – The result (fig. 4.18) of the question gives strong support to the findings in [38], that naming style does not impact the subjects' effort and program comprehension. Examining the results further there is a strong indication that the sample prefers camel case over snake case and pascal case. But since the participants could only leave one answer, and over 50% chose "It does not matter as long as we are consistent", we can not say for sure if these participants would have agreed if they were made to choose between the two.

Hungarian notation – Just over 60% of the participants answered that they think Hungarian notation has a negative impact on readability. This supports the findings of the literature. What is unexpected about the answers are the demographics of the participants who answered that they think it has a positive impact. The mean "years of experience in software development" of that group is 4.9 years less than the participants who answered that it has a negative impact. The expectation for this question was the opposite, that more experienced developers who started working with code before the intelligent development environments used today existed would be more prone to think Hungarian notation had a positive impact on readability. One possible explanation for this is that some of the less experienced participants were not familiar with Hungarian notation and answered without experience.

5.1.3 RQ3. Do IT-professionals experience that they always commit readable code? If not, what are the reasons for committing code that they do not consider as readable?

A brief summary of the answers given under the header "Reasons for committing "bad" code" in subsection 4.2.3:

Most participants feel satisfied with all or most code they commit to the projects they are working with. 67.17% answered somewhere on the scale from "more or less agree" to "strongly agree".

The most common reasons for committing code that the writer is not satisfied with are three answers all tightly connected. The one that the most people answered is unrealistic schedule. The following two are too small budget and lack of planning. If you have a tight budget and have not spent enough time on planning, you will probably end up with an unrealistic schedule.

Also, a group of participants answered that they do not commit code that they are fully satisfied with because "it is not worth the effort" or "good enough is sufficient".

5.1.4 RQ4. What countermeasures do IT professionals suggest to help them increase readability in code?

A brief summary of the answers given under the header "Activities for writing more readable code" in subsection 4.2.3:

82.8% of the participants answered that code review is an activity that helps write readable code. The answer frequency on pair programming, which is a closely related activity, was the third most frequent answer. A not too distant conclusion from this is that the participants of the study think that the best way to write readable code is to involve other people in some way.

The other popular answers were "more time" and usage of tools and software such as auto formatters and linters.

Chapter 6

Threats to validity

6.1 Not a representative sample

An active choice was made between choosing a small population and performing random sampling, such as one with one or more companies, IT students at BTH, or similar, or choosing a larger population and settling for a convenient sample.

Since the latter choice was chosen, the sample is not representative of the population and thus none of the conclusions or findings from the survey is valid as findings for the population.

6.2 Open invitations on social media

Most of the participants in the survey were reached by social media. The invitation was sent out to contacts and in groups/forums with a potential reach of about 20.000 people. Because the invitations were sent out open in groups and forums, without personal invitation (except a few invitations sent by email) the risk of decreasing representativity exists. Since the topic was revealed in the invitation, there is a possibility that people with a special-interest in the area could be more prone to take the survey than someone who does not care about it.

6.3 Not defining and describing concepts

All questions in the survey are short and concise with as little description as possible. This was an active choice to avoid that people do not finish the questionnaire. This might have come with the price of participants not understanding the questions or interpreting them differently.

6.4 Presumption of a shared definition of readability

The consequence of not defining readability to the respondents is possible problems in understanding the result. The respondents could hold a different definition of readability, something completely different than the definition used for gathering data in the literature review. This problem could easily have been avoided by adding a question asking the respondents about their definition of readability in a similar manner as with clean code.

6.5 Human error

Since the data and answers of the survey have been manually processed the possibility of human error in data processing exists. The risks has been minimized by rerunning all calculations and double checking all values.

6.6 Internal bias

The study might have been affected by internal bias of the author. Held beliefs by the author could have been passed on to questions and text of the study without intention and thus influencing the survey participants or readers of the study. To minimize this threat to validity the formulations were reviewed and piloted by others.

6.7 Sabotage & Hypothesis Guessing

Another possible threat is sabotage and hypothesis guessing, a risk that people with malicious intent answer the opposite of their opinions to sabotage the result or that the subject guesses the hypothesis and answers accordingly. The threat is considered unlikely, and the risks of this occurring were minimized by not indicating the intention to compare with academic results to the participants.

Chapter 7

Conclusions & Future Work

This study examined readability in source code, both in theory and in practice, why code with low readability is committed to projects, and what we can do to prevent it with a literature review and a survey. The result of the literature review presents 14 practices that increase or decrease readability in source code.

The findings of the survey revealed that, out of the presented practices, consistency is the most important practice when writing code to reach readability. Using names with full words found in dictionaries and avoiding long lines of code are two other practices supported by over 70% of the participants.

The result of the study suggests that the reasons for committing code with low readability mainly stem from lack of time and poor planning and time managing. Some participants explicitly expressed that it is not worth the effort to write perfect code.

The most suggested activities to help write readable code were code reviews, more time, and pair programming.

A possible future for continued studies is to present code snippets displaying code with and without the practices found in the literature review and to let the participants rate which solutions is the most readable. To add another dimension of results on readability, eye tracking could be used as a measurement. This could also be used to evaluate whether the participants experienced perception correlates to eye-tracking results and study if their beliefs in the practice are biasing their perception.

Another possible way to continue the research from this study is to apply data mining techniques to the data to find patterns that were not found in this study.

Bibliography

[1] "ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary," *ISO/IEC/IEEE 24765:2017(E)*, pp. 1–541, Aug. 2017, conference Name: ISO/IEC/IEEE 24765:2017(E).

[2] "readability," Jun. 2021, page Version ID: 62638303. [Online]. Available: https://en.wiktionary.org/w/index.php?title=readability&oldid=62638303

[3] "Definition of READABILITY," 2022. [Online]. Available: https://www.merriam-webster.com/dictionary/readability

[4] A. Almogahed and M. Omar, "Refactoring Techniques for Improving Software Quality: Practitioners' Perspectives," *Journal of Information and Communication Technology*, vol. 20, no. 4, pp. 511–539, 2021.

[5] E. Avidan and D. G. Feitelson, "Effects of Variable Names on Comprehension: An Empirical Study," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. Buenos Aires, Argentina: IEEE, May 2017. [Online]. Available: https://ieeexplore-ieee-org.miman.bib.bth.se/abstract/document/7961504

[6] B. Sharif and J. I. Maletic, "An Eye Tracking Study on camelCase and under_score Identifier Styles," in *2010 IEEE 18th International Conference on Program Comprehension*, Jul. 2010, pp. 196–205, journal Abbreviation: 2010 IEEE 18th International Conference on Program Comprehension.

[7] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Software Engineering*, vol. 18, no. 2, pp. 219–276, Apr. 2013. [Online]. Available: https://doi.org/10.1007/s10664-012-9201-4

[8] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To camelcase or under_score," in *2009 IEEE 17th International Conference on Program Comprehension*, May 2009, pp. 158–167, iSSN: 1092-8138.

[9] R. P. Buse and W. R. Weimer, "A metric for software readability," in *Proceedings of the 2008 international symposium on Software testing and analysis*, ser. ISSTA '08. New York, NY, USA: Association for Computing Machinery, Jul. 2008, pp. 121–130. [Online]. Available: http://doi.org/10.1145/1390630.1390647

[10] ——, "Learning a Metric for Code Readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, Jul. 2010, conference Name: IEEE Transactions on Software Engineering.

[11] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the Influence of Identifier Names on Code Quality: An Empirical Study," in *2010 14th European Conference on Software Maintenance and Reengineering*, Mar. 2010, pp. 156–165, iSSN: 1534-5351.

[12] J. Börstler and B. Paech, "The Role of Method Chains and Comments in Software Readability and Comprehension—An Experiment," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 886–898, Sep. 2016, conference Name: IEEE Transactions on Software Engineering.

[13] J. Börstler, H. Störrle, D. Toll, J. van Assema, R. Duran, S. Hooshangi, J. Jeuring, K. Hieke, C. Kleiner, and B. MacKellar, ""I know it when I see it" : perceptions of code quality." Association for Computing Machinery (ACM), 2017, pp. 389–389. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-70486

[14] E. Dale and J. S. Chall, "The Concept of Readability," *Elementary English*, vol. 26, no. 1, pp. 19–26, 1949, publisher: National Council of Teachers of English. [Online]. Available: http://www.jstor.org/stable/41383594

[15] E. W. Dijkstra, *Selected writings on computing : a personal perspective*. New York : Springer-Verlag, 1982. [Online]. Available: http://archive.org/details/selectedwritings0000dijk

[16] W. H. DuBay, *Smart Language: Readers, Readability, and the Grading of Text*, Jan. 2007, publication Title: Online Submission. [Online]. Available: https://eric.ed.gov/?id=ED506403

[17] 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)*, May 2019, pp. 2–12, iSSN: 2643-7171.

[18] B. J. Guzzetti, *Literacy in America: An Encyclopedia of History, Theory and Practice*. ABC-CLIO, 2002, google-Books-ID: rHNK8j_RrkkC.

[19] G. R. Klare, "measurement of readability," Iowa State University Press, Tech. Rep., 1963. [Online]. Available: https://scholar.google.com/scholar_lookup?title=measurement+of+readability&author=Klare%2C+George+Roger&publication_year=1963

[20] G. Lacerda, F. Petrillo, M. Pimenta, and Y. G. Guéhéneuc, "Code smells and refactoring: A tertiary systematic review of challenges and observations," *Journal of Systems and Software*, vol. 167, p. 110610, Sep. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220300881

[21] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a Name? A Study of Identifiers," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, Jun. 2006, pp. 3–12, iSSN: 1092-8138.

[22] ——, "Effective identifier names for comprehension and memory," *Innovations in Systems and Software Engineering*, vol. 3, no. 4, pp. 303–318, Dec. 2007. [Online]. Available: https://doi.org/10.1007/s11334-007-0031-2

[23] T. Lee, J.-B. Lee, and H. In, "A Study of Different Coding Styles Affecting Code Readability," *International Journal of Software Engineering and Its Applications*, vol. 7, pp. 413–422, Sep. 2013.

[24] ——, "Effect Analysis of Coding Convention Violations on Readability of Post-Delivered Code," *IEICE Transactions on Information and Systems*, vol. E98D, pp. 1–11, Jul. 2015.

[25] R. C. Martin, *Clean code : a handbook of agile software craftsmanship*. Upper Saddle River, N.J.: Prentice Hall, 2009.

[26] J. Pantiuchina, M. Lanza, and G. Bavota, "Improving Code: The (Mis) Perception of Quality Metrics," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2018, pp. 80–91, iSSN: 2576-3148.

[27] J. Pereira dos Reis, F. Brito e Abreu, G. de Figueiredo Carneiro, and C. Anslow, "Code Smells Detection and Visualization: A Systematic Literature Review," *Archives of Computational Methods in Engineering*, vol. 29, no. 1, pp. 47–94, 2022.

[28] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceedings of the 8th Working Conference on Mining Software Repositories*, ser. MSR '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 73–82. [Online]. Available: http://doi.org/10.1145/1985441.1985454

[29] T. Ribeiro and G. Travassos, "Attributes Influencing the Reading and Comprehension of Source Code – Discussing Contradictory Evidence," *CLEI Electronic Journal*, vol. 21, p. 5, Apr. 2018.

[30] ——, "Who is right? Evaluating empirical contradictions in the readability & comprehensibility of source code," 2017, pp. 99–112.

[31] C. Robson and K. McCartan, *Real world research : a resource for users of social research methods in applied settings*, fourth edition ed. Hoboken: Wiley, 2016.

[32] Y. Sasaki, Y. Higo, and S. Kusumoto, "Reordering Program Statements for Improving Readability," in *2013 17th European Conference on Software Maintenance and Reengineering*, Mar. 2013, pp. 361–364, iSSN: 1534-5351.

[33] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, and M. Beigl,

"Descriptive Compound Identifier Names Improve Source Code Comprehension," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, May 2018, pp. 31–3109, iSSN: 2643-7171.

[34] Y. Tashtoush, Z. Odat, I. Alsmadi, and M. Yatim, "Impact of Programming Features on Code Readability," *Computer Science Faculty Publications*, vol. 7, no. 6, Jan. 2013. [Online]. Available: https://digitalcommons.tamusa.edu/computer_faculty/16

[35] T. Tenny, "Program readability: procedures versus comments | IEEE Journals & Magazine | IEEE Xplore," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1271–1279, Sep. 1988. [Online]. Available: https://ieeexplore-ieee-org.miman.bib.bth.se/document/6171

[36] Y. Wang, "What motivate software engineers to refactor source code? evidences from professional developers," in *2009 IEEE International Conference on Software Maintenance*, Sep. 2009, pp. 413–416, iSSN: 1063-6773.

[37] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, "The effect of modularization and comments on program comprehension," in *Proceedings of the 5th international conference on Software engineering*, ser. ICSE '81. San Diego, California, USA: IEEE Press, Mar. 1981, pp. 215–223.

[38] Z. Sharafi, Z. Soh, Y. Guéhéneuc, and G. Antoniol, "Women and men — Different but equal: On the impact of identifier style on source code reading," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, Jun. 2012, pp. 27–36, journal Abbreviation: 2012 20th IEEE International Conference on Program Comprehension (ICPC).

Appendix A

Survey questions

1. **Age:** *integer*
2. **Gender:** *Female/Male/Other*
3. **Highest level of education:** *Secondary Education/Post-Secondary Education < 3 years/Post-Secondary Education >= 3 years/ PhD*
4. **Years of experience as a developer:** *Integer*
5. **Current occupation:** *Developer, Tester, Leader/manager, Teacher, Scientist, Student, Other*
6. **Role:** *mutiple checkboxes* Please select all that apply. As part of my job: I frequently write code. I frequently debug code written by others/legacy code. I frequently review others code.
7. **Programming best practices** *Please list if there are coding best practices mandated by your organization*
8. **What programming languages are you able to use proficiently?** *C/C++/Python/Java/C#/Visual Basic/JavaScript/PHP/SQL/GO/R/ Object Pascal/Apache Groovy/Ruby/Perl/MATLAB/Swift/Object-C*
9. **What programming languages do you use the most? proficiently?** *C/C++/Python/Java/C#/Visual Basic/JavaScript/PHP/SQL/GO/R/ Object Pascal/Apache Groovy/Ruby/Perl/MATLAB/Swift/Object-C*
10. **Are you aware of the term clean code?** *Bool*
11. **Define what clean code is to you.** *Textbox*
12. **For the following suggestions please indicate the extent to which you think they are likely to impact the readability and understandability of code:** *Strong negative impact/Negative impact/No impact/Positive impact/Strong positive impact*

(a) Length of lines of code (i.e. Occasionally writing long lines instead of breaking up a statement across multiple lines)

(b) Usage of numbers directly in the source code, opposed to first encapsulating it within a constant or a variable

(c) Naming conventions

i. Usage of Hungarian notation

ii. names with full words found in dictionaries

iii. Commonly used abbreviations such as "txt", "msg", "cnt" or "retVal"

iv. single letter identifiers

(d) Writing code in a format that is consistent with the previous code

(e) Usage of comments

(f) Nesting loops or if-statements

(g) Writing the "changing" variable in a condition on the left hand side of the comparison operator. (e.g., if (received_bytes < expected_byte){..)

(h) Usage of arithmetic formula directly in the source code, opposed to encapsulating it within a function.

(i) Defining all variables at the top of the method/function, opposed to defining the variable as close to the first reference as possible

13. **What naming style do you think are easiest to read?** *camelCase/-PascalCase/snake_case/Other/Does not matter*

14. **Do you think giving constant variables SCREAMING_SNAKE_CASE names increases readability?** *Bool*

15. **When I have completed a task and commit the code into a repository, I always feel that the code is refactored and written as readable and clean as I possibly can.** *Strongly disagree/Disagree/Neither agree or disagree/Agree/Strongly agree*

(a) *If not strongly agree:* **What are the reasons for committing code that you are not fully satisfied with?** *Lack of planning/Lack of interest/Lack of training or knowledge/Unrealistic schedule/To small budget/Other*

(b) **What activities does/would help you write readable and clean code** *Code reviews/pair programming/Education and seminars/More time/Better tools/other*