Bachelor's Thesis

CORRELATES OF CODE COMPLEXITY IN PROGRAM COMPREHENSION

YANNICK LEHMEN

November 21, 2022

Advisor:

Annabelle Bergum	Chair of Software Engineering
Norman Peitek	Chair of Software Engineering

Examiners:

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

Chair of Software Engineering
Saarland Informatics Campus
Saarland University

UNIVERSITÄT
DES
SAARLANDES

Erklärung

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

Statement

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

Einverständniserklärung

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

Declaration of Consent

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

Saarbrücken,_____ _____

(Datum/Date) (Unterschrift/Signature)

ABSTRACT

Code complexity metrics have seen widespread use to predict how hard comprehending a program is. However recent studies have cast some doubts on how well suited they actually are for this purpose. Subjective difficulty has been shown as a potentially better predictor then complexity metrics. In this thesis we will further investigate how strongly both complexity metrics and subjective difficulty correlate with the difficulty of program comprehension and the usage of different strategies for program comprehension. To this end we have used data from a previously conducted combined electroencephalogram (EEG) and eye-tracking study on program comprehension and searched for correlations with complexity metrics and subjective difficulty. In this the mental load measured by an EEG shows the difficulty of program comprehension while eye-tracking data helps us understand differences in comprehension strategy. We found that some complexity metrics can explain the difficulty of program comprehension pretty well. Subjective difficulty performed worse than multiple metrics but seems to correlate with an aspect of mental load that no complexity metric does. We also found significant differences in fixation length between iterative and recursive code snippets. Lastly, we found multiple relations that warrant a deeper investigation in future research.

CONTENTS

LIST OF FIGURES

1 INTRODUCTION

When a part of code in a larger project is very complex it can lead to a lot of problems in long-term development. Since complex code is difficult to understand it is harder to maintain, because a lot of tasks become more difficult. For example, it becomes harder for programmers to familiarize themselves with the code, which makes introducing new programmers to a project harder. In addition, maintenance becomes harder because errors are more likely to occur in more complex code [16]. Thus, it would be beneficial to rewrite code sections that are difficult to understand to be less complex and easier to understand.

However, to rewrite complex code sections we first need to be able to find these code sections. Using programmers' subjective judgement of difficulty is limited because it is a subjective measurement. This comes with two significant problems to its usability as a metric. Firstly, it is hard to automate and secondly as a subjective measurement it can differ to a significant extent between programmers. Instead metrics that use properties of the source code have been used to judge the complexity of code. Two such examples are lines of code and cyclomatic complexity. These metrics are objective and easy to automate.

However, it is still unclear how well complexity metrics correlate with the actual effort needed to understand a program. In the past years, a new wave of software engineering research has expanded the use of neuro-imaging techniques in software engineering research. This gives us the tools necessary to measure the mental load during program comprehension directly. Thus, we can use neuro-imaging techniques to validate if complexity metrics actually correlate with mental load. So far there only have been a few studies into this topic, with most of them being limited by a limited number of metrics and snippets used. A study by Peitek et al.[18] was the only study we found that used both a big set of snippets and numerous metrics. They found that overall subjective difficulty performed better then complexity metrics, with multiple widely used complexity metrics showing no or weak positive correlations with brain activation [18].

In this thesis we will use data from an experiment that used EEG to measure mental load. In the experiment eye-tracking and EEG data was collected for 37 participants. After the experiment participants were asked to give a subjective difficulty judgement for each snippet [17]. In addition, we calculated complexity metrics for each snippet where they were applicable. Using this data we will investigate correlations between code complexity (measured by complexity metrics and subjective difficulty) and both mental load and eye-tracking data.

,

2 BACKGROUND

In this chapter we will introduce necessary background information for this thesis.

2.1 PROGRAM COMPREHENSION

Program comprehension describes the family of tasks related to understanding the functionality of a source-code snippet. In every-day work basically every task of a programmer includes program comprehension [25]. Therefore, program comprehension tasks have been used in research for a long time to measure how difficult it is to work with a piece of code. The first big wave of research into program comprehension happened in the 1980s. This research did establish several theories regarding how programmers understand code, most importantly the difference between bottom-up and top-down comprehension [21]. Bottom-up comprehension describes a strategy in which a programmer reads a program line-by-line and constructs a mental model for the entire snippet based on the effect of each line. In contrast to bottom-up comprehension top-down comprehension describes a strategy in which a programmer starts out with a mental model of what the code does and checks if the rest of the code confirms this model [19]. The model can be based on telling identifiers, general structure, or other so-called beacons [1]. In general, the choice between using top-down and bottom-up comprehension is not a conscious decision but based on factors like experience and code clarity. These models for program comprehension strategies where mainly developed using techniques like think-aloud protocols or post-experiment interviews [19].

Siegmund [21] provided a general overview over the state of program comprehension research. This work outlined both the findings of the first wave of research into program comprehension and also more recent developments. After the first wave research into the human components of software engineering was largely ignored in favour of research into technical aspects. Only starting around 2010 did research into program comprehension gain steam again. This was driven by the usage of neuro-imaging techniques (see chapter 2.4) to gain a better understanding of the workings in the brain during program comprehension. This new wave of research is still ongoing, and here we present some important findings of these studies so far. A number of studies were important as they provided a proof-of-concept for using different neuro-imaging techniques in software engineering research [2, 3, 15, 22]. Another study showed the possibility of deciding code complexity based on mental load measurements [12]. Peitek et al. found correlations between programmer performance and mental load, with better programmers showing fewer and lower spikes in mental load during program comprehension tasks [17].

In addition, research using eye-trackers was conducted to further our understanding of theories about program comprehension strategies [20]. One of these studies found that

intermediate programmers read code less linearly then beginners and that the linearity of source code significantly impacts the linearity of reading the code [19]. Another study found that more proficient programmers are more likely to skip tokens, fixate code parts for a shorter amount of time and are less likely to revisit tokens. Overall, this shows a more efficient reading behaviour for more proficient programmers [17].

2.2 CODE COMPLEXITY

Code Complexity describes how hard to understand a source-code snippet is. Importantly, code complexity is not solely dictated by the functionality of the code but instead depends on the way this functionality is implemented. We can see that two source-code snippets with the same purpose can differ in terms of complexity by comparing a straightforward implementation of a function with one that includes a lot of unnecessary operations. The one with unnecessary operations is harder to understand, even if it implements the same functionality (see Figure 2.1).

```
void switch_values(int a, int b){
  if (a > b) {
    int help = a - b;
    b = a;
    a = b + help;
  }
  else {
    int help = b - a;
    a = b;
    b = a+help;
  }
}
```

```
void switch_values (int a, int b){
    int help = a;
    a = b;
    b = help;
}
```

Figure 2.1: Two methods that implement the same functionality (switching two values) but with different complexities. The right snippet simply switches two variables while the left one uses unnecessary complex operations

Overall, we would expect code complexity to correlate with the time needed to solve a task and with how often the task is solved successfully. For example, a more complex system would be harder to maintain, requiring more working time then a less complex system. Because of that a way to measure the complexity of a source-code snippet automatically could prove highly beneficial to software development by identifying parts of a program that might introduce problems or lead to higher maintenance effort when working on a program.

An intuitive way to measure code complexity is subjective difficulty, asking programmers working on it to judge how difficult the snippet is. Different studies found correlations between subjective difficulty ratings and answer correctness or time taken for a task [12, 18]. However, using subjective difficulty comes with the problem that there is no intuitive way to automatically classify source-code snippets using subjective difficulty. As such it is hardly usable in a practical environment because we want code complexity measurements

to be quick and automated. In addition, subjective difficulty is dependent on the individual programmer, because different people may judge the same snippet drastically different. For example, in the study used in the exploratory analysis in this thesis for six snippets ratings ranged from 1 (the easiest rating) to 10 (the hardest rating) and the ratings for a single snippet had a mean range of 7.94 values. Ideally, we would want an alternative to subjective difficulty that can be automated and is objective while still sharing similar correlations as subjective difficulty.

2.3 COMPLEXITY METRICS

One possibility for such an alternative are complexity metrics, which only depend on the actual program code (and comments in the code) and are thus objective and easy to automate. Complexity metrics have been used both in research and in practical development [18].

Some famous complexity metrics are McCabes cyclomatic complexity, Halsteads effort metric or Lines of Code. McCabes cyclomatic complexity metric calculates the amount of different paths through a program and is calculated by adding 1 to the number of if's, while's, for's, do's, switch cases, catches, conditional expressions, &&'s and ||'s in the method. When used in research and practical development, cyclomatic complexity is often used with a threshold at which methods are marked as too complex. A common threshold used is a cyclomatic complexity of 10 [18]. Lines of Code is a very basic metric that counts the amount of lines of code in a method. There are some different variants of this metric that count different lines as part of a metrics or not. In this thesis we used two different variants. One which included every line of code except whitespace in the method and one which also excluded comments. Halsteads effort metric is calculated from the number of distinct operators, the number of distinct operands and the total number of operands in the method [24].

Many metrics focus on a single aspect of code complexity. Cyclomatic complexity for example only considers the number of different paths through a program. Some other metrics combine multiple aspects, such as Halsteads effort metric, which should correspond with the level of difficulty of understanding a method. This in turn should intuitively be correlated to the difficulty during program comprehension. In addition, complexity metrics exist for various levels in a project hierarchy, from metrics that deal with a single method over methods over metrics on the level of classes and modules up to metrics for entire projects. Metrics on each of these levels can deal with different aspects of code complexity. As an example, cyclomatic complexity captures the control-flow of a single method, while module metrics often deal with the connections between different modules.

While complexity metrics are widely used there is insufficient evidence that they actually are a good representation of code complexity. If they were we would expect them to correlate with higher difficulty of program comprehension. Peitek et al. [18] found that subjective difficulty correlated stronger with mental load (see Chapter 2.4) than any of the complexity metrics they used. In addition, no single metric currently used captures all

aspects of code complexity [9].

2.4 MENTAL LOAD

Mental load is a psychological term that describes the amount of work the brain has to do to solve a task. When measuring mental load for a task we want to specifically measure mental activity caused by this task and isolate that from the mental load caused by background activity.

Neuro-imaging techniques are methods we can use to capture brain activity, either by measuring it directly or by measuring effects caused by brain activity. The main techniques used to measure brain activity are EEG, functional magnetic resonance imaging (fMRI) and functional near-infrared spectroscopy (fNIRS). FMRI and fNIRS both measure the flow of oxygenated blood to activated brain areas. To do so they use electromagnetic fields (fMRI) or infrared light (fNIRS), both of which react differently depending on the oxygen level of blood. While this response can be located very well, especially when using fMRI, it takes some time after the onset of brain activity to occur and is thus hard to match with a specific stimulus. As such fMRI has high spatial resolution, meaning that which brain areas are active can be identified very well, and low temporal resolution, meaning that we cannot pinpoint when a brain area is activated or deactivated [6]. In contrast, EEG directly measures the electronic potentials created by brain activity.
Aside from these techniques there also are some less common ways to measure mental load, for example heart rate variability (HRV) measurements. These measure the increase in heart rate caused by increased mental activity. Different neuro-imaging techniques all have different strengths and weaknesses that lead to different experimental designs and means they capture different aspects of mental load [2]. We will describe EEG in more detail, because it was used in the study we used for this thesis.

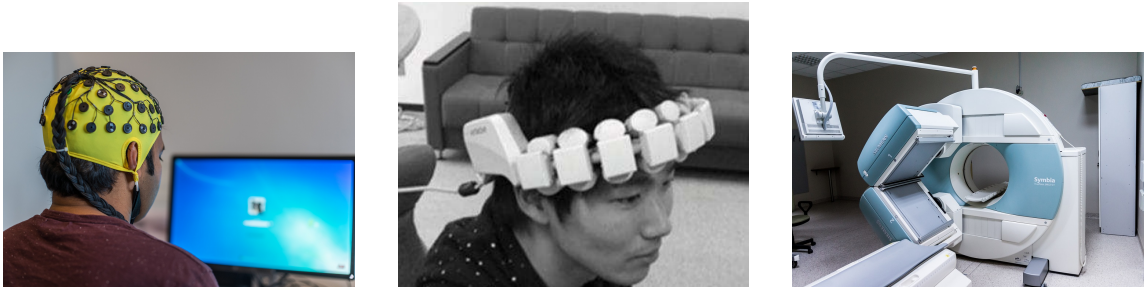

Figure 2.2: Images of different neuro-imaging techniques, form left to right: An EEG cap in use [27], a measurement tool used in fNIRS [15] and a MRI device as used for fMRI [28].

EEG is a non-invasive neuro-imaging device that can be used to measure the electrical potentials created by mental activity in the brain. To use an EEG electrodes are placed on the participants skull and a conductive gel is used to connect them to the skin so that electrical potentials can be measured. While the participant performs a task each of these electrodes continuously measures the electrical potential at the place it is attached to. This allows EEG

to have a very high temporal resolution, because changes in electrical potentials happen within milliseconds of a stimuli appearing. However, EEG has a very poor spatial resolution because it is mathematically not possible to discern where the electrical potentials originate based only on the values present at the electrodes. As a result, it is not possible to calculate which brain areas are active based only on EEG measurements. It is however possible to use EEG to check an assumption about where in the brain a process occurred. Such an assumption could for example be based on prior work using a technique with good spatial resolution like fMRI. [11]

Program comprehension tasks are tasks used in software engineering research to measure how difficult it is to work with a piece of code. These tasks are appropriate for this purpose because program comprehension features in nearly every task a programmer might face. Thus, program comprehension tasks enable us to measure mental load during understanding a source-code snippet. Some of the earliest program comprehension tasks used in research were modifying a source-code snippet to fit a given specification [4] and rewriting code from memory [4, 21]. These tasks were all assumed to be easier the better the program was understood based on prior psychological research. In contrast, newer studies instead used tasks more directly related to program comprehension, such as first understanding a snippet and then giving the output for a given input [18].

Ideally, we would expect that measurements of code complexity correlate with mental load during program comprehension. So, we would expect a more complex source-code snippet to require more mental effort to understand then a simpler snippet. If this assumption holds then mental load during program comprehension tasks should strongly correlated to code complexity metrics. In practice these assumptions could so far not be proven, for example a study by Peitek et al. [18] found at best weak to medium correlations between mental load and code complexity metrics.

3

RELATED WORK

In the previously mentioned study, Peitek et al. [18] conducted an experiment in which they used fMRI to measure mental workload. The snippets they used were similar to ours, small Java snippets without identifying names, but all of the snippets only consisted of one method. They searched for correlations between complexity metrics and mental workload as well as subjective difficulty and mental workload. Peitek et al. found small to medium correlations for complexity metrics and mental workload but a strong correlation for mental workload and subjective difficulty.

A number of studies analysed the correlations between mental workload and program comprehension. A study using fMRI to measure mental workload during top-down and bottom-up comprehension of Java source-code snippets was conducted by Siegmund et al. [23]. In the study participants were shown snippets designed to facilitate or hinder top-down comprehension. They found a significantly lower mental workload for snippets enabling top-down comprehension than for snippets that necessitate bottom-up comprehension. In addition, they identified a brain area that was activated for bottom-up comprehension and deactivated for top-down comprehension. Medeiros et al. [12] conducted a study in which they measured mental workload using EEG. They used three different Java source-code snippets and collected subjective difficulty ratings from the participants. They were able to build a classifier that could identify whether data belonged to the easiest source-code snippet with both high recall and precision. However, when data belonged to one of the two more difficult snippets the classifier was not able to distinguish between them in a reliable manner. This also matched up with the subjective difficulty ratings in which the two more complex snippets scored nearly equal. Of note is that the two snippets had significantly different complexity metric values for multiple metrics, most notable cyclomatic complexity, but neither mental workload nor subjective difficulty reflected that.

Yeh et al. [26] conducted a study in which they used EEG to measure mental workload during code comprehension tasks on C/C++ snippets. For each snippet they used two versions that computed the same result. One of the versions was intentionally changed to be more confusing by obfuscating identifiers and introducing statements with no effect on the result. They found that mental workload was higher for confusing source-code snippets and also that mental workload correlated with correctness of the answers.

Overall, these studies all found links between mental workload and subjective difficulty or mental workload and code complexity. They found that mental workload can be used to distinguish between snippets seen as easy or hard according to subjective difficulty. However, the ability to distinguish between snippets based on mental workload in these studies was somewhat limited, only being able to distinguish between snippets with clear differences in subjective difficulty.

Aside from studies dealing with relations between mental workload and program comprehension there also exist other studies in the field of software engineering using different techniques to measure mental workload. Nakagawa et al. [15] used fNIRS to measure the cerebral blood flow while comprehending easy and hard C source-code snippets using bottom-up strategies. Similar to the study by Yeh et al. the hard snippets were created by obfuscating easy snippets. They found increased blood flow for hard tasks, aligning with the expected higher mental load for these tasks. This suggests that fNIRS is a viable way to measure mental workload for programming. Couceiro et al. [2] used HRV as a measurement of mental workload during program comprehension tasks on Java snippets. They found that it matched the subjective difficulty experienced by the participants much closer then cyclomatic complexity did. Both HRV and subjective difficulty were nearly equal for code snippets with highly different cyclomatic complexity. Kosti et al. [10] used EEG to measure mental workload during program comprehension tasks on snippets in the programming language C. They were able to construct models that distinguished between syntax tasks and program comprehension tasks based on the matching EEG data. Overall, these studies have shown that different techniques to measure mental workload are suitable in software engineering research.

In addition to this more recent research into program comprehension and code complexity there also exist a number of older studies into code complexity. These often-found correlations between code complexity and programmers' performance in program comprehension tasks or subjective difficulty. For example, Curtis et al. [4] conducted an experiment were participants were shown Fortran snippets and then had to recall the program or modify it in a specified way. They found that both Halsteads effort metric and McCabe's cyclomatic complexity correlate with programmer's performance in these tasks. In another study Kafura et al. [7] searched a long-term developed system for complexity outliers. They then asked developers heavily involved in the project for a subjective difficulty evaluation of the different parts of the project. They showed that for a number of complexity metrics the outliers were parts of the program associated with problems or confusion during development. In a more recent study Katzmarki et al. [9] matched programmers' subjective difficulty rankings of Java snippets with complexity metrics for those snippets. They found that while there is some correlation between complexity metrics and subjective difficulty no metric fully agrees with subjective difficulty and also that more complex metrics did not necessarily outperform simpler metrics that measure the same general aspects of a program.

In the study that is also used for this thesis Peitek et al. [17] used EEG and eye-tracking to collect data from participants during program comprehension tasks. They also collected experience measurements for the participants. They found that more efficient programmers read code more efficiently and have a lower mental workload during comprehension. They also found that some common experience measures like years of programming did not correlate with efficiency, while some less common measures such as self-estimation and measurements of learning eagerness correlated with efficiency.

In general, the connection between complexity metrics and mental workload has only

been explored a little. Most studies that used complexity metrics did not deal with mental workload and most studies that used mental workload used heuristics or subjective judgement for code complexity. When studies included complexity metrics it is often only in a small manner including only a few complexity metrics. A counterexample to this is the fMRI study by Peitek et al. which did include a high number of complexity metrics and also found some weak to medium correlations between complexity metrics and mental workload. However, with it only being a single study the generalisability of their findings is limited. As such to what extent currently used complexity metrics actually correlate with code complexity and mental workload during code comprehension is still not fully clear and we aim to expand upon the existing research into the topic by using another technique to measure mental workload. In regard to the other main aspect of our study, the correlation between eye movement and complexity metrics, we were not able to find any previous work that dealt with the effect code complexity has on eye movement and reading strategies.

4

STUDY DESIGN

In the following we will describe the overall design of our study. We will begin by describing the experiment conducted by Peitek et al. [17]. Then we will present our research questions and explain how we will use the data from the experiment by Peitek et al. for our analysis.

4.1 EXPERIMENT DESIGN

The experiment was conducted by Peitek et al. for a study dealing with programming experience and performance [17]. They had participants solve program comprehension tasks while recording their eye movements using an eye tracker and using an EEG to measure brain waves. We will use the data collected in this experiment for this thesis.

For the experiment 39 participants were invited. Every participant had at least one year of experience with Java or three years of experience with a related language (for example C#). Of those participants one did not complete the experiment and the data of one participant was not included in the analysis because they were an outlier in regard to time taken to answer, being too slow for more than half of the tasks. For one of the remaining 37 participants the eye-tracker could not be calibrated, so only EEG data was collected for them. As such EEG data from 37 and eye-tracking data for 36 participants were available for our analysis. In the experiment participants had to complete program comprehension tasks. One task

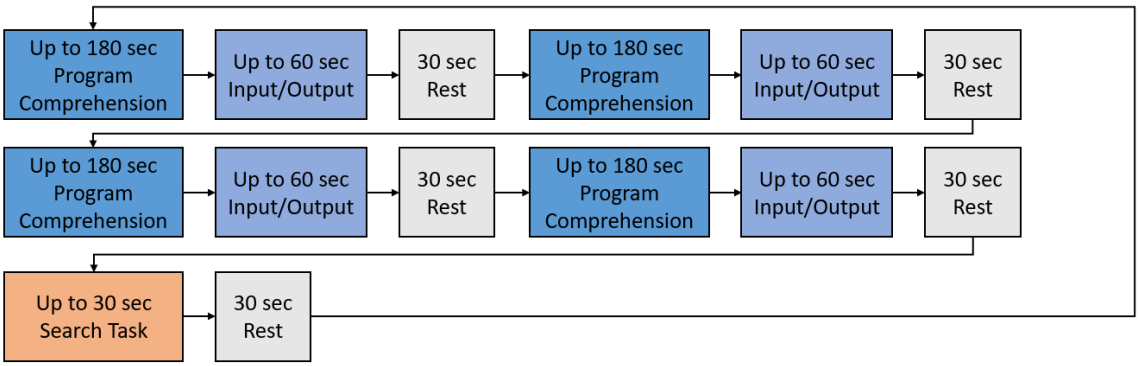

Figure 4.1: The order of the different tasks during the experiment.

consisted of two steps. In the first step the participants were shown a source-code snippet until they confirmed they understood the snippet or 180 seconds passed. Afterwards the snippet was replaced by an input and the participants had to calculate the output for this input. When they confirmed that they had solved the task they were shown four possible outputs and an option to skip the task. The selection of the correct output was limited to 60 seconds. The participants were asked to select the output the snippet would give for the input or select skip if they weren't able or willing to solve the task. As a distraction task

for the EEG measurements a search task in which participants had to count brackets was included. The distraction and comprehension task were ordered so that participants always completed blocks of four comprehension and one distraction tasks. Between tasks a rest condition in the form of cross fixation was included. Each rest condition lasted 30 seconds. Figure 4.1 presents the overall order of tasks in the experiment.

The experiment lasted up to an hour, with breaks of up to five-minute length after 20 and 40 minutes. In total up to 32 program comprehension tasks were presented. Due to the experiment length being limited to an hour most participants answered less than 32 tasks.

After the experiment each participant answered a postquestionnaire which included them ranking the difficulty of each task. This subjective difficulty rating was performed by participants ranking all the snippets on a scale from 1 to 10 using papers with images of the snippets on them. Each participant was only handed snippets they actually saw during the experiment. In some cases, a participant could not remember that they saw a snippet. In this case the snippet was excluded from the rating.

4.2 RESEARCH QUESTIONS

We will use the data from this experiment to answer a number of research questions. As we explained in the section regarding related work, while many studies exist dealing with code complexity metrics and also an ever-growing number of studies using mental load during program comprehension studies the correlations between code complexity and mental load are still largely unexplored. As such the first research question we focus on is:

RQ1: What correlations between complexity metrics and mental load during program comprehension tasks exist?

In addition to mental load data the experiment also collected eye-tracking data which could give us insight into different usage of strategies to comprehend source-code snippets of different difficulties. As such our second research question is:

RQ2: What correlations between complexity metrics and eye-tracking metrics during program comprehension tasks exist?

The first study in the area of correlations between code complexity and mental load found that overall subjective difficulty is often correlated more strongly with mental load then with complexity metrics [18]. We want to continue this avenue of research by studying if there is a metric or multiple metrics that correlate with subjective difficulty particularly strongly. If we find such a metric it might be usable as a replacement for subjective difficulty that can easily be automated. As such our third research question is:

RQ3: What correlations between complexity metrics and subjective difficulty ratings for methods exist?

Lastly, we want to see how subjective difficulty performs as a method for judging mental load. In addition, we want to look at the connections between subjective difficulty and eye-tracking. As such our fourth and fifth research questions are:

RQ4: What correlations between subjective difficulty and mental load during a program comprehension task exist?

RQ5: What correlations between subjective difficulty and eye movement during a program comprehension task exist?

4.3 DATA

In this section we will explain how we obtained and analysed the data for our study.

4.3.1 *Complexity Metrics*

The experiment by Peitek et al. did not use complexity metrics so we had to find an appropriate set of metrics to use for our study. One problem we found with this was that the source-code snippets in the experiment were not designed with applying a set of metrics to them in mind. Most of the snippets presented to the participants consist of only one method, which would make method complexity metrics an appropriate option. However, a number of snippets consist of multiple methods or include helper classes. For these snippets method complexity levels are not able to capture the entirety of the snippet in their measurement. To have a set of complexity metrics that can be used for every snippet we would thus need to use module complexity metrics. Using module complexity metrics would come with other disadvantages which we believe would outweigh the advantage of using more snippets. Thus, we decided to use method complexity metrics. In the following we will explain advantages and disadvantages of both module and method complexity metrics for our data set.

The first aspect, which we already mentioned is the number of snippets the metric can be used for. All 32 snippets consisted of a single module and thus could be measured with module complexity metrics. For method complexity metrics six snippets consist of more than one method and thus could not be appropriately measured with method complexity metrics and one snippet had a helper class that would not be taken into account by the method complexity metrics. Overall, this means using module complexity metrics would allow us to use seven additional snippets for our study.

However if we use module complexity metrics we would face two large issues, firstly we could not use many commonly used metrics such as cyclomatic complexity, because these are only defined for methods and not for modules. In addition, while some metrics that are not typically used at a module level could easily be adopted for use at such a level, some metrics, again including the commonly used cyclomatic complexity, cannot be cleanly used on a module level. While in theory these important method level metrics could be replaced by other often used module level metrics, such as ones concerned with connections with other modules, basically all of these metrics would be unusable, because they are

designed to judge a module that is part of a bigger project, often in terms of connections with other modules. Because all modules presented in the experiment work completely isolated from each other we could not apply these metrics in a sensible manner.

As such we decided to use method metrics, because being able to use more metrics and metrics that are more meaningful for the snippets was more important than using more snippets with less meaningful metrics. In theory using class level complexity metrics was also possible, however these would have shared all the issues module complexity metrics had while still excluding one snippet and thus this option was not further investigated.

After deciding to use method complexity metrics we then had to find a fitting collection of metrics to use. We used the MetricsReloaded Plugin for IntelliJ [14] to calculate all available method level metrics. We then removed all metrics whose values where always zero for all of our snippets. This removed eleven out of 41 metrics. Then in a second pass we checked if there were any two metrics with exactly the same values for each snippet and found two such pairs. Thus, we removed one metric from each of these pairs from our analysis, to prevent duplicate values. A list of all initially used metrics with reasons for excluding the metric if we didn't use it in our analysis can be found in the appendix (Section A.1).

4.3.2 *Mental Load*

To measure mental load, we used the mental load data provided by the original study. The data was calculated using the ratio of alpha to theta band measurements. Since theta band activity mostly represents background activity while the brain is idle and alpha band activity mostly represents active thought, like it is required to comprehend program code, this gives a good measurement of mental load. In addition to the study by Peitek et al. [17] this measurement has been shown by Medeiros et al. [13] to be a good measurement of mental load in computer science research. It is also commonly used in psychological work, for example by Holm et al. and Kartali et al. [5, 8] Since we use the same metric as the study by Peitek et al. we reused their already calculated mental load values instead of recalculating them for our study. The measurements include a data point every 0.1 seconds, with a single data point being comprised of values from a 3 second sliding window. This gives us values over time for each participant and each snippet.

From these values we determined outliers by calculating the inter-quartile range for each snippet and removing every value further away from the median than 3 times the inner-quartile range. If more than 50% of the values of a participant for a single algorithm were marked as outliers we removed the entire measurement as an outlier, otherwise we kept the entire measurement. Initially we tried to also remove individual data points if they were marked as outliers, however this would have removed a lot of valid data due to the waveform of mental load data (See Figure 4.2 for why outlier removal was necessary).
We used 3 times the interquartile range instead of the more common 1.5 times the inner quartile range to make up for the true high peaks in mental load data we used (See Figure 4.3). If more than half of the measurements for a single participant were removed we

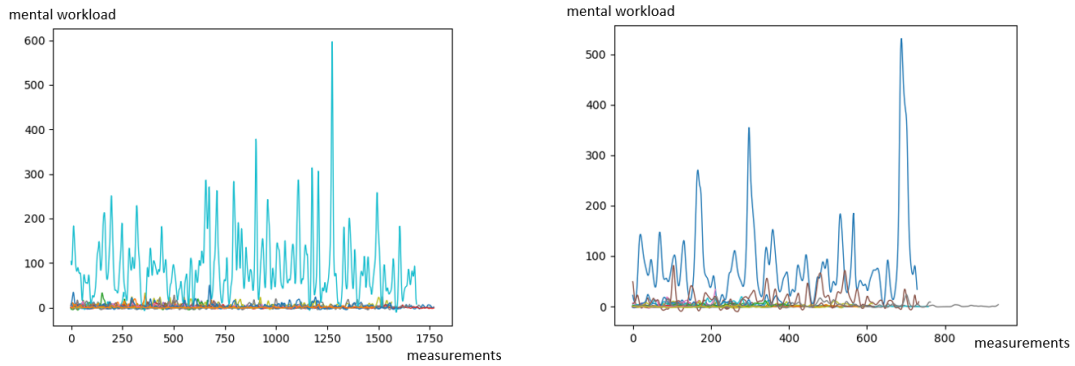

Figure 4.2: Examples of our mental load data for two tasks before outlier removal (RabbitTortoise on the left and MedianOnSorted on the right). Each colored line represents data from one participant. It shows clearly that data for at least one participant is a significant outlier in each of these tasks. Similar outliers existed for most tasks for a few participants. If we did not remove these outliers our mental load measurements, especially the maximum mental load would have been heavily distorted.

excluded the entire participant to account for potential errors in the EEG set-up that could render these measurements unusable.

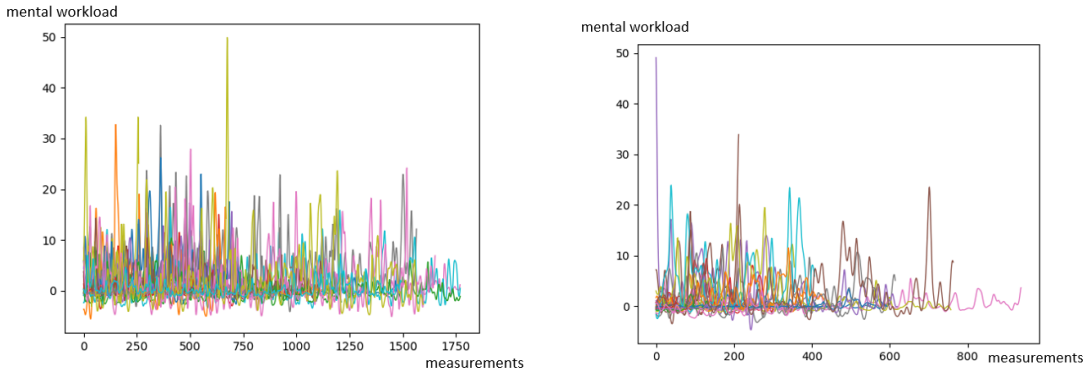

Figure 4.3: The data for the snippets in Figure 4 (RabbitTortoise on the left and MedianOnSorted on the right) after removing outliers (more then half of values outside of 3 times the inter-quartile range). The upper bound of 1.5 times the inter-quartile range was below 10 for each snippets. As we can see using 1.5 times the inter-quartile range we would have cut off a lot of data due to the peaks in mental load data.

We then calculated three different values for each algorithm and each participant. These values were the maximum and minimum of the mental work values and the mean of all values. To obtain a single value for each snippet we then calculated the mean for each of the values. This yielded three metrics we could use for our analysis: MW-min, MW-max and MW-mean.

1. MW-max: The mean of the maximum mental load during a task, calculated over all participants

2. MW-mean: The mean of the mean mental load during a task, calculated over all participants
3. MW-min: The mean of the minimum mental load during a task, calculated over all participants

4.3.3 *Eye Tracking Metrics*

For our eye-tracking metrics we obtained data in the form of fixation lengths for all fixations during the experiment. The data was divided into first fixations and refixations and available on three different levels, tokens, areas of interest (AOI) and lines of code. We chose to only use data from the token and AOI level, because lines of code would be less granular then tokens and less telling then AOI. An example for how AOIs and tokens were marked can be seen in Figure 4.4. For both levels we then calculated the following metrics: The average fixation length for first fixations, refixations and all fixations, the average percentage of fixations that were refixations and the percentage of tokens/AOIs that were skipped. We counted a token as skipped if it was never fixated. The fixation length data from the original experiment was divided into first fixations and refixations. However in some cases a token or AOI had two lengths for first fixations given. This happened when a participant fixated a token or AOI for the first time, then the eye-tracker lost track of their fixation and the next fixation was again the same token or AOI. Possible reasons for this could be that the participant looked off-screen, which would disrupt the eye-tracking or that the eye-tracker failed to track the participants eyes for some time, for example because of a glare on their glasses. We decided to only count the first of these fixations as a first fixation and every subsequent one as a refixation. We did this because we can't know for certain if there was a fixation in between those two fixations or not. In total this yielded ten eye-tracking metrics we used in our analysis. A list of all used eye-tracking metrics can be found in the appendix (Section A.2).

4.3.4 *Subjective Difficulty*

For the subjective difficulty ratings of the snippets, we used ratings obtained in a post-questionnaire in the original study. For this, participants were asked to sort all of the snippets regarding how difficult they were on a scale from one to ten. These values exist for 38 out of 39 participants because one participant did not finish the post-questionnaire. To obtain single values for each algorithm we then calculated the median subjective difficulty of each snippet (SD-Median). This is necessary for our analysis because we compare subjective difficulty and complexity metrics. Thus, we need to bring our subjective difficulty data into a form that is comparable to that of complexity metrics, one value per algorithm.

Instead of the median we could also have used the mean of the subjective difficulty ratings of a snippet. However, we do not know if the participants used the scale from 1 to 10 as a linear scale or in some other fashion. For example, in the postquestionnaire a participant explained that they used the scale with small differences between 1 to 6 and increasingly

Figure 4.4: This graphic showcases the selection of AOIs and tokens for the example of the BogoSort snippet. On the left the tokens are marked, encompassing singular words or characters. On the right the AOIs are marked, encompassing bigger logical areas of the program.

bigger differences between 7 to 10. Because the mean assumes a linear scale, we decided to use the median.

5 RESULTS

In this section we will present the results of our analysis. To check for correlations between our metrics we used kendalls-tau. We decided to use kendalls-tau (instead of pearson's correlation coefficient) due to it dealing better with outliers and being well suited for ordinal and continuous data. The Thresholds we used correlations can be seen in Figure 5.1.

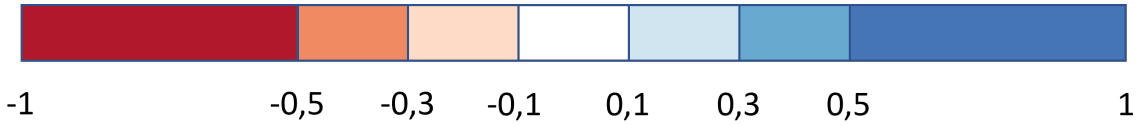

Figure 5.1: This graph shows the different categories of correlation we used. The graph reaches from strong negative correlations to strong positive correlations.

5.1 RESEARCH QUESTION 1

In the following we will present the results relevant to research question 1:
What correlations between complexity metrics of methods and mental workload during Program comprehension tasks exist?

All correlations between mental workload and complexity metrics can be found in Figure 5.2. For MW-mean we found mostly no or weak positive correlations. However, CALLED and RETURN showed weak negative correlations. MW-min shows no or a weak negative correlation with most metrics. We also found a weak positive correlation with the metrics ev(g) and IF_NEST. In Addition, BRANCH showed a medium positive correlation with MW-min. For MW-max we found weak to medium positive correlations with most complexity metrics. Four metrics (BRANCH, CALL, CALLED and ev(G), while D and E showed strong positive correlations with MW-max.

The metrics that showed medium or strong positive correlations with MW-max are a diverse set of metrics. Both very simple metrics (LOC) and more complicated metrics (QCP metrics) are part of that group. These metrics also measure different aspects of the program, for example control flow metrics (v(G)), vocabulary metrics (N) and code size metrics (STAT) all show medium positive correlations.

	BRANCH	CALL	CALLED	CAST	CDENS	CONTROL	D
MW-max	0,093	0,065	-0,026	0,118	0,134	0,365	0,509
MW-mean	0,031	0,011	-0,170	0,165	0,051	0,112	0,246
MW-min	0,303	-0,049	-0,119	-0,259	0,093	0,067	-0,132

	E	ev(G)	EXEC	EXP	IF_NEST	iv(G)	LOC
MW-max	0,507	0,039	0,322	0,466	0,161	0,246	0,409
MW-mean	0,260	0,078	0,112	0,223	0,085	0,138	0,129
MW-min	-0,080	0,156	-0,077	-0,108	0,275	-0,130	0,024

	LOOP	LOOP_NEST	N	n	NBD	NP	QCP_CRCT
MW-max	0,393	0,278	0,467	0,269	0,142	0,063	0,483
MW-mean	0,360	0,313	0,225	0,310	0,117	-0,143	0,195
MW-min	-0,103	0,013	-0,084	0,003	0,229	-0,045	-0,067

	QCP_MAINT	QCP_RLBTY	RETURN	RLOC	STAT	V	v(G)
MW-max	0,455	0,462	-0,130	0,155	0,368	0,384	0,398
MW-mean	0,194	0,199	-0,112	0,135	0,165	0,177	0,190
MW-min	-0,087	-0,064	-0,067	0,041	-0,018	-0,037	0,004

Figure 5.2: This table shows the correlations between our mental workload measurements and the complexity metrics we used.

Overall, we found that maximum and mean mental workload correlate with a wide array of different metrics, but none of the metrics on their own explain mental workload. Maximum mental workload showed significantly stronger correlations then mean mental workload. For minimum mental workload we found at best weak correlations and no clear trend regarding positive or negative correlations.

5.2 RESEARCH QUESTION 2

In the following we will present the results relevant to research question 2:
What correlations between eye-tracking metrics and complexity metrics during program comprehension tasks exist?
The correlations between eye-tracking metrics using tokens and complexity metrics can be found in Figure 5.3. The correlations between eye-tracking metrics using AOIs and complexity metrics can be found in Figure 5.4.

For the different fixation length metrics, we found mostly weak or no correlations with a few exceptions. Multiple metrics showed medium negative correlation with one or more fixation length metrics. For the CALLED metric we also found a medium positive correlation for the length of refixations on tokens (L_rf-TKN).

For the remaining metrics we will deal with those using tokens first. For both the likelihood of a token being skipped (P_skip-TKN) and of a fixation being a refixation (P_rf-TKN) we found mostly no or weak positive correlations. A few metrics also showed weak negative correlations (P_skip-TKN with BRANCH and P_rf-TKN with CAST and RETURN).

	BRANCH	CALL	CALLED	CAST	CDENS	CONTROL	D
L_ff-TKN	-0,056	-0,186	0,139	-0,047	0,058	-0,239	-0,037
L_fix-TKN	-0,068	-0,027	0,284	-0,071	0,216	-0,022	0,044
L_rf-TKN	-0,068	0,072	0,325	-0,024	0,223	0,007	0,051
P_skip-TKN	-0,155	0,065	-0,026	0,000	0,010	0,030	0,024
P_rf-TKN	0,241	0,011	0,057	-0,165	0,120	0,127	0,111

	E	ev(G)	EXEC	EXP	IF_NEST	iv(G)	LOC
L_ff-TKN	-0,153	0,078	-0,378	-0,108	-0,123	-0,085	-0,269
L_fix-TKN	-0,020	0,094	-0,287	-0,020	0,076	-0,013	-0,052
L_rf-TKN	-0,020	0,117	-0,294	-0,034	0,123	0,058	-0,010
P_skip-TKN	0,113	0,086	-0,035	0,128	0,038	0,013	-0,024
P_rf-TKN	0,033	0,055	0,070	0,027	0,171	0,058	0,136

	LOOP	LOOP_NEST	N	n	NBD	NP	QCP_CRCT
L_ff-TKN	-0,252	-0,234	-0,124	-0,140	-0,263	-0,045	-0,154
L_fix-TKN	-0,409	-0,402	0,010	-0,187	-0,246	0,260	-0,013
L_rf-TKN	-0,426	-0,419	-0,003	-0,194	-0,220	0,296	-0,020
P_skip-TKN	-0,087	-0,084	0,084	0,119	-0,065	0,081	0,007
P_rf-TKN	0,136	0,102	0,084	0,024	0,134	0,009	0,134

	QCP_MAINT	QCP_RLBTY	RETURN	RLOC	STAT	V	v(G)
L_ff-TKN	-0,167	-0,212	0,219	-0,088	-0,361	-0,144	-0,294
L_fix-TKN	-0,027	-0,057	0,255	0,020	-0,242	-0,023	-0,078
L_rf-TKN	-0,040	-0,064	0,290	0,007	-0,228	-0,023	-0,033
P_skip-TKN	0,054	0,051	0,121	0,101	0,046	0,104	0,108
P_rf-TKN	0,120	0,132	-0,103	-0,027	0,046	0,037	0,100

Figure 5.3: This table shows the correlations between our eye-tracking metrics using tokens and the complexity metrics we used.

	BRANCH	CALL	CALLED	CAST	CDENS	CONTROL	D
L_ff-AOI	-0,130	-0,042	0,191	0,283	-0,072	-0,261	-0,064
L_fix-AOI	-0,080	-0,141	0,201	-0,047	0,038	-0,089	0,091
L_rf-AOI	-0,105	-0,125	0,222	-0,118	0,065	-0,089	0,051
P_skip-AOI	0,006	0,460	0,212	0,094	0,120	0,164	0,165
P_rf-AOI	0,390	-0,148	-0,160	-0,165	0,134	0,350	0,307

	E	ev(G)	EXEC	EXP	IF_NEST	iv(G)	LOC
L_ff-AOI	-0,067	-0,008	-0,259	-0,142	-0,085	-0,058	-0,157
L_fix-AOI	-0,013	-0,047	-0,217	0,061	-0,104	-0,121	-0,066
L_rf-AOI	-0,040	-0,039	-0,245	0,034	-0,095	-0,121	-0,094
P_skip-AOI	0,180	0,219	0,077	0,149	0,266	0,219	0,178
P_rf-AOI	0,287	0,031	0,385	0,291	0,152	0,022	0,311

	LOOP	LOOP_NEST	N	n	NBD	NP	QCP_CRCT
L_ff-AOI	-0,260	-0,225	-0,158	-0,051	-0,272	0,170	-0,168
L_fix-AOI	-0,343	-0,375	0,050	-0,290	-0,375	0,215	0,027
L_rf-AOI	-0,360	-0,393	0,024	-0,303	-0,367	0,215	0,007
P_skip-AOI	-0,079	-0,066	0,138	0,160	0,082	0,054	0,134
P_rf-AOI	0,368	0,340	0,319	0,133	0,272	0,027	0,362

	QCP_MAINT	QCP_RLBTY	RETURN	RLOC	STAT	V	v(G)
L_ff-AOI	-0,181	-0,199	0,165	-0,392	-0,277	-0,157	-0,294
L_fix-AOI	0,020	-0,024	0,094	0,034	-0,207	-0,017	-0,152
L_rf-AOI	-0,007	-0,051	0,138	0,047	-0,235	-0,043	-0,152
P_skip-AOI	0,127	0,125	0,246	-0,054	0,109	0,164	0,182
P_rf-AOI	0,361	0,388	-0,389	0,176	0,361	0,290	0,294

Figure 5.4: This table shows the correlations between our eye-tracking metrics using AOIs and the complexity metrics we used.

For the metrics using AOIs we found the likelihood of an AOI being skipped (P_skip-AOI) to show weak positive or no correlations with the complexity metrics. The only Exception is CALL which has a medium positive correlation with P_skip-AOI. The likelihood of a fixation being a refixation (P_rf-AOI) shows mostly weak or medium positive correlations. A few metrics showed no (NP, ev(g), iv(G)) or a weak negative (CALL, CALLED, CAST) correlation. In addition, RETURN shows a medium negative correlation.

The weak correlations for the probability of a fixation on a token being a refixation (P_rf-TKN) and of a token being skipped (P_skip-TKN) would suggest that complexity metrics are not able to capture what causes decision making for eye-movements on a token level. On the other hand, the same metrics for AOIs showed overall stronger correlations, suggesting complexity metrics are better suited to predict eye-movements on an AOI level than on a token level.

With a few exceptions only weak correlations between fixation length metrics and complexity metrics exist. The likelihood of a token being skipped or refixated showed only at most weak correlations. The likelihood of an AOI being skipped also showed mostly only weak correlations. The likelihood that a fixation on an AOI was a refixation showed medium correlations with multiple metrics.

5.3 RESEARCH QUESTION 3

In the following we will present the results relevant to research question 3:
What correlations between complexity metrics and subjective difficulty ratings for methods exist?

	BRANCH	CALL	CALLED	CAST	CDENS	CONTROL	D
SD Median	0,180	0,139	0,239	-0,177	0,218	0,356	0,380

	E	ev(G)	EXEC	EXP	IF_NEST	iv(G)	LOC
SD Median	0,294	0,092	0,237	0,276	0,219	0,168	0,345

	LOOP	LOOP_NEST	N	n	NBD	NP	QCP_CRCT
SD Median	0,053	-0,033	0,325	0,007	0,130	0,221	0,393

	QCP_MAINT	QCP_RLBTY	RETURN	RLOC	STAT	V	v(G)
SD Median	0,331	0,344	0,010	0,091	0,230	0,240	0,232

Figure 5.5: This table shows the correlations between subjective difficulty and complexity metrics

All correlations between subjective difficulty and complexity metrics can be seen in Figure 5.5.

Overall, we found mostly weak and medium correlations. Six metrics showed no correlation (LOOP, LOOP_NEST, ev(G), n, RETURN and RLOC). In addition, for the CAST metric we found a weak negative correlation.

The metrics that showed medium positive correlations were a mix of simple metrics (like LOC) and more complex metrics (like QCP_RLBTY). Notably every metrics that showed a medium correlation here also has a medium or strong correlation with maximum mental workload (see Chapter 5.5). The only negative correlation we found was for the metric CAST. However, because CAST only has a value different from zero for one of our snippets it is possible that the correlation would change with a set of snippets that is more balanced regarding the metric.

Complexity metrics did show some positive correlations with subjective difficulty. QCP_CRCT and D showed the strongest correlations. The only negative correlation was for CAST and probably caused by limits of our data set. Multiple metrics also had no correlation with subjective difficulty.

5.4 RESEARCH QUESTION 4

In the following we will present the results relevant to research question 4:
What correlations between subjective difficulty and mental workload during a program comprehension task exist?

The correlations we found for mental workload and subjective difficulty can be found in Figure 5.6. We can see that the correlation strongly depends on what aspect of mental workload we look at. For MW-max we found a medium positive correlation, meaning that for snippets with higher subjective difficulty ratings maximum mental workload was higher. For MW-mean we found no correlation with subjective difficult, despite multiple

	MW-max	MW-mean	MW-min
SD Median	0,365	-0,057	-0,301

Figure 5.6: This table shows the correlations between subjective difficulty and mental workload

	L_ff-TKN	L_fix-TKN	L_rf-TKN	P_skip-TKN	P_rf-TKN
SD Median	0,079	0,193	0,165	-0,501	0,559

	L_ff-AOI	L_fix-AOI	L_rf-AOI	P_skip-AOI	P_rf-AOI
SD Median	-0,243	0,229	0,215	-0,172	0,508

Figure 5.7: This table shows the correlations between subjective difficulty and eye tracking

complexity metrics showing medium or weak correlations with MW-mean. MW-medium showed a medium negative correlation with subjective difficulty, meaning that for snippets with higher subjective difficulty the minimum mental workload was lower. This is notably the strongest negative correlation we found for minimum mental workload.

Subjective difficulty showed a medium positive correlation with maximum mental workload, no correlation with mean mental workload and a medium negative correlation with minimum mental workload.

5.5 RESEARCH QUESTION 5

In the following we will present the results relevant to research question 5:
What correlations between subjective difficulty and eye movement during a program comprehension task exist?

All correlations between eye-tracking metrics and subjective difficulty can be seen in Figure 5.7. We found weak positive correlations with subjective difficulty for the lengths of refixations and all fixations. The length of first fixations showed weak negative correlation with subjective difficulty for AOIs and no correlation for tokens.

The probability of a fixation being a refixation showed strong positive correlations with subjective difficulty for both tokens and AOIs. This suggests that how often they go back to already fixated parts of a program significantly impacts a programmer's assessment of difficulty.

The probability of an AOI being skipped showed a weak negative correlation with subjective difficulty while the probability of a token being skipped showed a strong negative correlation. The difference between AOIs and tokens here might be explained by the size of AOIs, with them being larger it is significantly less likely for an AOI to be skipped and the correlation is thus influenced more easily by outliers or the way AOIs were defined.

Fixation length metrics showed at most weak correlations with subjective difficulty. The likelihood of a fixation being a refixation showed strong positive correlations with subjective difficulty for both tokens and AOIs. The likelihood of a token being skipped showed a strong correlation with subjective difficulty while the likelihood of an AOI being skipped only showed a weak correlation.

6 DISCUSSION

In this section we will discuss interesting correlations and provide possible interpretations as well as tie in related literature

6.1 COMPLEXITY METRICS

The strongest correlations for MW-max were found with D and E, both metrics that were designed specifically to correlate with the effort or difficulty of designing and understanding a program and seem to fulfil that purpose well on our data.

The strongest correlations for MW-mean were with LOOP and LOOP_NEST, with LOOP_NEST being one of only two metrics to correlate stronger with MW-min then MW-max. A possible reason for this might be that the Expressions used inside a loop in the snippets were often not complex, being simple arithmetic operations. Instead, the complexity here comes from repeatedly doing the same operations, something which we would expect to take a longer relatively stable amount of effort instead of single peaks. Notably our data set has only values from 0 to 2 for LOOP_NEST and from 0 to 4 for LOOP. This could mean our findings regarding the LOOP and LOOP_NEST metrics do not necessarily hold for data sets with a wider spread of values. An experiment with a set of snippets specifically designed to contain different LOOP and LOOP_NEST values could find clearer results here.
Notably we found a medium positive correlation of cyclomatic complexity (v(G)) with MW-max and a weak positive correlation with MW-mean. This stands in contrast to prior research using fMRI by Peitek et al. [18] in which they found no correlations between v(G) and any of the measures of brain activation or deactivation they used. However, it is important to note that their experiment did not include a measurement representing maximum mental workload where we found the stronger correlation. In difference to MW-mean and the fMRI measurements used by Peitek et al. MW-max does not show effort over an extended period of time but only a peak of mental load during the task. For example, a snippet that is generally easy but has a difficult part could lead to a low MW-mean and a high MW-max. In contrast a snippet that is constantly difficult would have a relatively high MW-mean but a low MW-high in comparison to other snippets. The correlation with MW-mean might be explained by differences in the snippets we used. Overall, we cannot confirm the findings by Peitek et al. that cyclomatic complexity shows no correlations with mental workload. We believe further research is necessary to conclusively answer the question, especially given that both of the study by Peitek et al. and the experiment we used for our data did not include snippets with cyclomatic complexity above 10, a common threshold for warning about cyclomatic complexity being to high [18]. Such research could focus on singular metrics or limited sets of metrics. This would enable us to use sets of snippets designed to have a spread of values for these metrics while ruling out other sources of complexity by

keeping other metrics as stable as possible.

For minimum mental workload we found mostly weak correlations, suggesting it might not be captured well by complexity metrics. The only exception is the BRANCH metric, which showed a medium positive correlation.

For fixation length metrics the strongest negative correlations notably were with metrics that directly or indirectly indicate an iterative method (LOOP, LOOP_NEST and NBD). The strongest positive correlations were with metrics indicating a recursive method (RETURN and CALLED). This might mean that there is a significant difference in reading strategy for iterative and recursive functions. A possible explanation for this is that the statements within loops in our data tend to be relatively easy executable statements which participants look at repeatedly to understand the loop. This might lead to a series of short fixations. Future research could use a qualitative analysis of eye-tracking data to possibly show this difference or find a different explanation for the differences in fixation length between recursive and iterative functions.

6.2 SUBJECTIVE DIFFICULTY

Subjective difficulty correlated weaker with maximum mental workload then 12 of the complexity metrics we used. For medium mental workload we found an at least weak positive correlation for twenty complexity metrics but none for SD-MEDIAN. This stands in strong contrast to findings by Peitek et al. [18], which found medium to strong correlations between subjective difficulty and their measurements of mental workload and had subjective difficulty correlate stronger with mental workload then most metrics. The reason for this difference is unclear so far but it might be caused by differences in the used neuro-imaging techniques and measurements of mental workload. The study by Peitek et al. used fMRI and used the deactivation of default mode network brain areas while we used EEG and the ratio between alpha and theta power bands.

The medium negative correlation we found between MW-min and subjective difficulty is also notable, because it is significantly stronger than any correlation, we found for complexity metrics and minimum mental workload or for eye tracking metrics and minimum mental workload. This indicates that there might be some aspect of mental workload that the different complexity metrics and eye-tracking metrics we used did not capture that subjective difficulty does capture.

Overall, these results suggest that subjective difficulty could probably not be used to replace complexity metrics, because it performs significantly worse than many of them regarding the correlation with mental workload. However there seems to be some element of subjective difficulty that is relevant to mental workload but not captured by any of the complexity metrics we used. Finding what aspect exactly that is and why it seems connected with minimum mental workload requires further investigation. Notably it is also

possible for a change in behaviour regarding very hard tasks to explain the lower MW-min. This would mean that participants disengaged from tasks they found particularly hard, explaining the difference in MW-min. Future research could further investigate the relation between minimum mental workload and subjective difficulty. The first step to do so would be to exclude the possibility of disengagement as a confounding factor. For this a qualitative analysis of individual participants behaviour during tasks they judge as difficult would be well suited. If such a study would find that disengagement is unlikely to cause the relation we found than further investigation into the topic should continue.

For fixation length we found no to weak correlations with subjective difficulty. The probability that a token gets skipped showed a strong negative correlation with subjective difficulty. Notably for AOIs the probability of one getting skipped only showed a weak correlation with Subjective difficulty. This points towards programmers skipping singular tokens when a snippet is easy for them to interpret. This is coherent with the idea of top-down comprehension, with programmers forming an assumption about the snippet and skipping tokens when they already have a strong assumption what should be in that position. Future research shall explore using a qualitative analysis of eye-tracking data to show in more detail which tokens are skipped.

Both for tokens and AOIs the probability of a fixation being a refixation correlates strongly with subjective difficulty. This means that how often a programmer goes back to an already fixated part of the code impacts their assessment of how difficult code is to comprehend. Future research shall attempt to use these eye-tracking metrics to predict subjective difficulty. If this is indeed possible it would allow us to predict subjective difficulty solely based on objective data, removing one of the two big hurdles towards using subjective difficulty as a practical measurement of complexity.

7 THREATS TO VALIDITY

In the following we will outline some possible threats to the validity of our results. We will divide this section between threats to the internal validity of our results and threats to the external validity of our results.

7.1 THREATS TO INTERNAL VALIDITY

As we used data from an experiment by Peitek et al. [17] threats to the internal validity of that experiment also apply to our study in so far as they are relevant to the data we used. Specifically, this means that by using data from the experiment we also adopted the operationalization of program comprehension inherent to that experiment. This operationalization divides program comprehension into a process with multiple steps, first comprehending the code then calculating the output for a given input and lastly selecting the output from multiple options. Given the complexity of program comprehension this certainly is not the only possible operationalization of program comprehension, but the one used here was specifically designed to make sure participants actually try to comprehend the snippet [17]. One possible drawback of this approach is that participants might guess an option when they calculated the wrong output, and their solution was not one of the options. To minimize this risk the experiment had the option to skip answering if participants were not sure of their answer.

To avoid fatigue effects the experiment was ended after an hour. To make sure that this does not cause large disparities in the amount of data available the order of snippets was randomized [17].

As previously discussed, the correlation we found between SD-Median and MW-min might not point towards a unique aspect of mental workload captured by subjective difficulty but instead could be explained by participants disengaging from particularly difficult tasks.

7.2 THREATS TO EXTERNAL VALIDITY

The generalizability of our results is limited by a number of factors. Firstly our study was limited only to relatively small snippets in Java. As such our results may not be generalizable to larger programs or to programs in other programming languages. Additionally we limited our analysis only to method level metrics due to them being suited best to analyse our snippets. This however further limits the generalizability to larger problems, because these method metrics are not cleanly applicable to larger programs.

Regarding our overall verdict on complexity metrics, while we used a wide selection of metrics it is possible that we missed metrics that would have correlated differently with some or all of our other metrics. This is an issue we can not fully eliminate, due to the high and constantly growing amount of complexity metrics. Lastly among the metrics we used there are some that only showed a limited span of values on our snippets. This might mean that results for some of these metrics are not generally applicable because our snippets do not vary enough regarding the metrics. One example of this would be cyclomatic complexity, for which our snippets only included values from two to seven. However a value of ten is often considered a threshold for a warning regarding high complexity, so none of our snippets actually exceeded that threshold. Another example is the CAST metric, which had a value of one for a single snippet and a value of zero for all other snippets. As such results using the CAST metric are very limited regarding their generalizability.

More in-depth studies for a single metric or a small set of metrics would be able to design sets of snippets specifically to cover a wide range regarding that metric. For our study we instead decided to focus on getting data for as many complexity metrics as possible. We believe this to be a good approach to extend the so far very limited pool of existing results regarding the relation between complexity metrics and mental workload.

8 CONCLUSION

Complexity metrics are an important tool for researchers and programmers, but their actual correlation with comprehension strategies and mental load is still unclear. We conducted an analysis into the correlations between complexity metrics and both eye-tracking and mental load. For this analysis we used subjective difficulty, eye-tracking and EEG data from a pre-existing experiment and calculated complexity metrics for each of the snippets used. Due to our usage of method metrics, we were limited to snippets consisting of a single method.

For maximum mental workload we found mostly weak to medium correlations with complexity metrics. Mean mental workload showed overall weaker correlations then maximum mental workload. We especially found that Halsteads effort and difficulty metrics correlated strongly with mental load. For minimum mental workload we found mostly insignificant correlations. For fixation length metrics we found a significant difference between metrics indicating recursive methods and metrics indicating iterative methods.

For subjective difficulty we found a medium correlation with maximum mental workload and no correlation with mean mental workload. In contrast to previous research this means that subjective difficulty overall showed weaker correlations with mental load then most complexity metrics. Notably we also found a medium negative correlation between subjective difficulty and minimum mental workload, potentially pointing to an aspect of mental load captured by subjective difficulty that none of the complexity metrics we used can explain.

For our eye-tracking metrics the probability of a fixation being a refixation showed a strong positive correlation with subjective difficulty. The probability of a token being skipped showed a strong negative correlation with subjective difficulty.

We found some supporting evidence that complexity metrics do properly represent at least part of the mental load during program comprehension. However, given the limited generalisability of our study we do not believe our findings are sufficient to give clear advice for practical applications. To clearly determine if complexity metrics can approximate the difficulty of program comprehension in a practical environment future research is needed. This research could for example focus on a single complexity metric for which we found promising results and use snippets specifically designed to cover a wide range of values for this metric. This would allow a more focused investigation of the relation between this complexity metric and mental load.

The difference in fixation length between recursive and iterative functions is another interesting finding that future research shall elaborate on to deepen our understanding of

different program comprehension strategies.

Another possibility we found is the relation between eye-tracking metrics and subjective difficulty. Specifically the probability of a token being skipped and of a fixation being a refixation showed strong correlations with subjective difficulty. This could mean that these are a possible way to predict subjective difficulty based on objective data.

APPENDIX

A.1 LIST OF COMPLEXITY METRICS

The following is a list of all complexity metrics that were used in the experiment.

1. ASSERT: The number of assertions in the method. This metrics was discarded, because its value was always 0, no snippet used assertions.

2. B: Halsteads bug metric, intended as an estimate of the number of bugs in a function. It is calculated as $B = V/3000$, where V is Halsteads volume metric. This metric was excluded because it was 0 for all our snippets.

3. BRANCH: The number of non-structured branch statements in a method. Non-structured branches include continue statements and branch statements outside of switch statements.

4. CALL: The number of method calls in a method.

5. CALLED: The number of places in the project from which the method may be called.

6. CALLEDp: The number of places in the product code of the project from which the method may be called. This has the same value as CALLED for each method and was thus excluded.

7. CAST: The number of typecast or instanceof expressions in the method. This method has the value 0 for all but one method. Thus we used it but we can only make limited conclusions regarding this metric based on such a limited set of snippets

8. CAUGHT: The number of expression classes caught in the method. This metric was zero for all snippets and was thus excluded.

9. CDENS: The ratio of control statements to all statements in the method.

10. CLOC: The amount of lines of comments in the method. The value of this metric was zero for all snippets and it was thus excluded.

11. COM_RAT: The ratio of lines of comments to total lines of code in the method. This metric was zero for all snippets and was thus discarded.

12. CONTROL: The total number of control statements in a method.

13. D: Halsteads difficulty metric, intended to correspond to the level of difficulty of understanding or programming a method. It is calculated as $(\eta_1/2) * (N_2/\eta_2)$, where eta_1 is the number of distinct operators in the method, eta_2 is the number of distinct operands in the method and N_2 is the total number of operands in the method.

14. E: Halsteads effort metric, intended to correspond to the level of effort necessary to maintain a method. It is calculated as $D * V$, where D is Halsteads difficulty metric and V is Halsteads volume metric.

15. ev(G): The essential complexity of a method. This is a graph-theoretic measurement of how ill-structured the control flow of a metric is. This reaches from 1 to v(G), the cyclomatic complexity of a method.

16. EXEC: The total number of executable statements in a method.

17. EXP: The total number of expressions in a method.

18. IF_NEST: The highest nesting depth of conditional statements in a method

19. iv(G): The design complexity of a method. This is a measurement of how interlinked a methods control flow i with calls to other methods. This reaches from 1 to v(G), the cyclomatic complexity of a method.

20. JLOC: The lines of javadoc comments in the method. We did not comment the snippets and thus this metric was zero for all snippets and was excluded.

21. LOC: The lines of code in a method. This includes comments but excludes whitespace.

22. LOOP: The number of loop statements in a method.

23. LOOP_NEST: The maximum nesting depth of loops in a method.

24. N: Halsteads length metric, it is the total number of operands and operators in a method. It is calculated as $N = N_1 + N_2$, where N_1 is the total number of operators and N_2 is the total number of operands.

25. n: Halsteads vocabulary metric, it is the total number of distinct operands and operators in a method. It is calculated as $n = \eta_1 + \eta_2$, where η_1 is the number of distinct operators and η_2 is the number of distinct operands in a method.

26. NBD: The maximum nesting depth of a method.

27. NCLOC: The number of lines of code in a method. This excludes whitespaces and comments. Because we did not use any comments this metric has the same value as LOC for each snippet and was thus excluded.

28. NP: The number of parameters in a method.

29. NTP: The total number of type parameters in a method. This is zero for all snippets and thus was excluded.

30. NULL: The number of comparisons with null in a method. This is zero for all snippets and was thus excluded.

31. QCP_CRCT: The Quality Control Profile metric for correctness. This is designed to determine the correctness of a method. It is calculated as $QCP\_CRCT = D + CONTROL + EXEC + (2 * v(G))$.

32. QCP_MAINT: The Quality Control Profile metric for maintainability. This is designed to estimate the difficulty of maintaining a method. It is calculated as $QCP\_MAINT = (3 * N) + EXEC + CONTROL + NEST + (2 * v(G)) + BRANCH$

33. QCP_RLBTY: The Quality Control Profile metric for reliability. This is designed to estimate the reliability of a method. It is calculated as $QCP\_RLBTY = N + (2 * NEST) + (3 * v(G)) + BRANCH + CONTROL + EXEC$

34. RETURN: The number of return points in a method.

35. RLOC: The ratio of lines of code of a method to the lines of code in its class.

36. STAT: The number of statements in a method.

37. TCOM_RAT: The ratio of lines of comments to lines of source code in a method. This is always zero for our snippets and was thus excluded.

38. THROWS: The number of exception classes a method is marked as throwing. This is zero for our snippets and was thus excluded.

39. TODO: The number of TODO comments in a method. This is zero for all our snippets and was thus excluded.

40. V: Halsteads volume metric, intended to correspond to the size of a method. It is caclulated as $V = N * log(n)$, where N is the Halstead length metric and n is the Halstead vocabulary metric.

41. v(g): The cyclomatic complexity of a metric.

A.2 LIST OF EYE-TRACKING METRICS

The following is a list of all eye-tracking metrics we used in our analysis:

1. L_fix-TKN: The mean length of all fixations on tokens.
2. L_ff-TKN: The mean length of all first fixation on tokens.
3. L_rf-TKN: The mean length of all fixations on tokens after the first for each tokens.
4. P_skip-TKN: The probability that a token was never fixated by a participant.
5. P_rf-TKN: The probability of a fixation being a refixation on token level.
6. L_fix-AOI: The mean length of all fixations on AOIs.
7. L_ff-AOI: The mean length of all first fixation on AOIs.
8. L_rf-AOI: The mean length of all fixations on AOI after the first for each AOI.
9. P_skip-AOI: The probability that a AOI was never fixated by a participant.
10. P_rf-AOI: The probability of a fixation on an AOI being a refixation.

BIBLIOGRAPHY

[1] Ruven Brooks. "Towards a theory of the comprehension of computer programs." In: *International Journal of Man-Machine Studies* 18.6 (1983), pp. 543–554.

[2] Ricardo Couceiro, Goncalo Duerte, Joao Duraes, Joao Castelhano, Catarina Duerte, César Teixeira, Miguel Castelo Branco, Paulo de Cravalho, and Henrique Madeira. "Biofeedback Augmented Software Engineering: Monitoring of Programmers' Mental Effort." In: *Proceedings of 41st ISCE-NIER*. IEEE, 2019, pp. 37–40.

[3] Igor Crk, Timothy Kluthe, and Andreas Stefik. "Understanding Programming Expertise: An Empirical Study of Phasic Brain Wave Changes." In: *ACM Transactions on Computer Human Interaction* 23 (2015).

[4] Bill Curtis, Sylvia B. Sheppard, Phil Milliman, M.A. Borst, and Tom Love. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics." In: *Transactions on Software Engineering* SE-5 (1979), pp. 96–104.

[5] Anu Holm, Kristian Lukander adn Jussi Korpela, Mikael Sallinen, and Kiti Müller. "Estimating Brain Load from the EEG." In: *The Scientific World Journal* 9 (Feb. 2009), pp. 639–651.

[6] Scott Hüttel, Allen Song, and Gregory McCarthy. *Functional Magnetic Resonance Imaging*. Sinauer Associates, 2014.

[7] Dennis Kafura and Geereddy R. Reddy. "The Use of Software Complexity Metrics in Software Maintenance." In: *Transactions of Software Engineering* SE-13 (1987), pp. 335–343.

[8] Aneta Kartali, Milica Janković, Ivan Gligorijevic, Pavle Mijovic, Bogdan Mijovic, and Maria Leva. "Real-Time Mental Workload Estimation Using EEG." In: Oct. 2019, pp. 20–34.

[9] Bernhard Katzmarski and Rainer Koschke. "Program Complexity Metrics and Programmer Opinions." In: *Proceedings of 20th IEEE*. ICPC, 2012, pp. 17–26.

[10] Makrina Viola Kosti, Kostas Georgiadis, Dimitrios A. Adamos, Nikos Laskaris, Diomidis Spinellis, and Lefteris Angelis. "Towards and affordable brain computer interface for the assessment of programmers' mental workload." In: *International Journal of Human-Computer Studies* 115 (2018), pp. 52–66.

[11] Steven Luck. *An Introduction to the Event-Related Potential Technique*. 2014.

[12] Júlio Maedeiros, Ricardo Couceiro, Joao Castelhano, Miguel Castelo Branco, Goncalo Duerte, Catarina Duerte, Joao Duraes, Henrique Madeira, Paulo Carvalho, and César Teixeira. "Software code complexity assesment using EEG features." In: *Proceedings of 41st EMBC*. IEEE, 2019, pp. 1413–1416.

[13] Julio Medeiros, Ricardo Couceiro, Goncalo Duerte, Joao Duraes, Joao Castelhano, Catarina Duarte, Miguel Castelo-Branco, Henrique Madeira, Paulo de Carvalho, and Cesar Teixeira. "Cann EEG Be Adopted as a Neuroscience Reference for Assessing Software Programmers' Cognitive Load." In: *Sensors* 21 (2021).

[14] *MetricsReloaded Plugin*. URL: https://plugins.jetbrains.com/plugin/93-metricsreloaded/ (visited on 10/29/2022).

[15] Takao Nakagawa, Yasutaka Kamei, Hidetake Uwano, Akito Monden, Kenichi Matsumato, and Daniel M. German. "Program Comprehension: Past, Present, Future." In: *Companion Proceedings of the 36th ICSE*. ACM, 2014, pp. 448–451.

[16] Edward E. Ogheneovo. "On the Relationship between Software Complexity and Maintenance Costs." In: *Journal of Computer and Communications* 2 (2014), pp. 1–16.

[17] Norman Peitek. "Correlates of Programmer Efficacy and their Link to Experience: A Combined EEG and Eye-Tracking Study." In: *ACM 2022 too appear*. IEEE/ACM, 2022.

[18] Norman Peitek, Sven Apel, Chris Parnin, André Brechmann, and Janet Siegmund. "Program Comprehension and Code Complexity Metrics: An FMRI Study." In: *Proceedings of 43rd ICSE*. IEEE/ACM, 2021, pp. 524–536.

[19] Norman Peitek, Janet Siegmund, and Sven Apel. "What Drives the Reading Order of Programmers? An Eye Tracking Study." In: *Proceedings of ICPC '20*. ACM, 2020, pp. 342–353.

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

[21] Janet Siegmund. "Program Comprehension: Past, Present, Future." In: *Proceedings of 23rd SANER*. IEEE, 2016, pp. 13–20.

[22] Janet Siegmund, Christian Kästner, Sven Apel, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. "Understanding understanding soure code with functional magnetic resonance imaging." In: *Proceedings of ICSE*. ACM Press, 2014, pp. 378–389.

[23] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. "Measuring Neural Efficiency of Program Comprehension." In: *Proceedings of ESEC/FSE '17*. ACM, 2017, pp. 140–150.

[24] Chandrasegar Thirumalai, Shridharshan R R, and Ranjith Reynold L. "An Assessment of Halstead and COCOMO Model for Effort Estimation." In: *Proceedings of i-PACT 2017*. 2017, pp. 1–4.

[25] Rebecca Tiarks. "What Maintenance Programmers Really Do: An Observational Study." In: *Workshop on software reengineering*. Citeseer. 2011, pp. 36–37.

[26] Martin K.-C. Yeh, Dan Gopstein, Yu Yan, and Yanyan Zhuang. "Detecting and Comparing Brain Activity in Short Program Comprehension Using EEG." In: *2017 IEEE FIE Conference*. IEEE, 2017, pp. 1–5.

[27] *pixabay*. URL: https://pixabay.com/de/photos/eeg-integration-hirnstrommessung-2680957/ (visited on 10/25/2022).

[28] *pixabay*. URL: https://pixabay.com/de/photos/mrt-kernspintomographie-diagnose-2813899/ (visited on 10/25/2022).