
UNIVERSIDADE D
COIMBRA

José Pedro Luz de Sousa Cruz

# iREVIEW: AN INTELLIGENT TOOL FOR CODE REVIEW QUALITY EVALUATION USING BIOFEEDBACK

**Dissertation in the context of the Master in Informatics Engineering, specialization in Software Engineering, advised by Eng. Haytham Hijazi and co-advidsed by Professor Henrique Madeira and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.**

September of 2022

UNIVERSITY OF COIMBRA

MASTER THESIS

# iReview: an Intelligent Tool for Code Review Quality Evaluation using Biofeedback

*Author:*
José Cruz

*Supervisor:*
Haytham Hijazi
*Co-advisor:*
Prof. Henrique Madeira

*A thesis submitted in fulfillment of the requirements for the degree of Master in Software Engineering*

Department of Informatics Engineering
CISUC - Centre for Informatics and Systems of the University of Coimbra

September 5, 2022

# Declaration of Authorship

I, José Cruz, declare that this thesis titled, "iReview: an Intelligent Tool for Code Review Quality Evaluation using Biofeedback" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

*"We can only see a short distance ahead, but we can see plenty there that needs to be done."*

Alan Turing

# *Abstract*

Code Review is a powerful tool to ensure high software quality by detecting bugs and providing feedback. Producing high-quality software requires productive developers and attentive code reviewers to ensure the quality of their work. Studies show that developers produce 70 bugs for every 1000 lines of code, and 75% of a developer's time is spent on debugging. Yet, how can we ensure a good code review quality before the code is finally released? Modern code review became more lightweight and individual reliant (does not need structured group meetings). Typically, one reviewer performs code reviews using platforms and tools like GitHub and GitLab. However, these tools do not evaluate the code review quality based on an individual's attributes (e.g., cognitive state). Changes in the reviewer's cognitive state due to difficulties in understanding code under review or stress and distraction might affect the quality of the review. Therefore, this thesis introduces an intelligent tool that evaluates the code review quality by using non-intrusive biofeedback devices (i.e., smartwatches) that assess the code reviewer's cognitive state (the engagement level). The tool uses a desktop eye-tracker, compatible with the software development environment, to identify regions where the code regions have not been well-reviewed, thus advising a second review. Biometric features extracted from smartwatches such as Heart Rate Variability (HRV) and Electrodermal Activity (EDA) and other features such as complexity of the code under review, review time, and the number of revisits to that region are the input of the Machine Learning (ML) model. The ML model classifies each code region as either well or badly reviewed with a pointer to the badly reviewed code regions. This tool is expected to help software companies achieve better code review quality and train new code reviewers on best code review practices.

# *Resumo*

A revisão de código é um processo muito importante na garantia de alta qualidade de um Software. Estudos indicam que em cada 1000 linhas de código escritas por desenvolvedores de software, existam 70 bugs e que em média 75% do tempo investido em desenvolvimento é utilizado a realizar debugging de código. Desenvolver um bom software requer uma boa equipa de desenvolvedores produtivos e uma boa equipa de revisionadores para garantir a qualidade do software. Ainda assim, como podemos garantir que os testes de qualidade de software são realizados com qualidade antes do software ser lançado? Estudos anteriores mostram várias ligações entre características biométricas e o estado emocional e cognitivo em tempo real, que permite a recolha de informações acerca das diversas emoções e estados cognitivos pelas quais o indivíduo em causa experiencia. Sendo as revisões de código a última barreira anterior ao lançamento do produto, estes desempenham um papel muito importante na prevenção de bugs do mesmo. Assim, o processo de revisão de código essencial para a garantia de qualidade do software.

O processo de revisão consiste na análise individual do código desenvolvido. Sendo a revisão em si tão dependente da pessoa que a está a rever, tornou-se assim mais "leve" e dependente, no sentido em que o revisor que a faz pode fazê-lo no seu tempo sem a necessidade de se reunir com outros revisores ou membros da equipa. Como podemos assim, assegurar que o revisor está nas suas melhores condições mentais e cognitivas para poder assegurar qualidade na sua revisão? Para este propósito, introduzimos sensores não-intrusivos capazes de fornecer feedback acerca das características biométricas identificadas no revisor durante a revisão, para que seja possível medir a sua dedicação (a nível de atenção, distrações, dificuldades etc) durante a tarefa. Nesta tese utilizamos um eye-tracker que permite obter a região para onde o revisor está a olhar, associando assim determinadas características biométricas a diferentes regiões do código e um smartwatch, que nos permite obter leituras de outras características biométricas.

Utilizando essas mesmas características biométricas (i.e. HRV, EDA e eye-tracking) e não-biométricas (i.e. complexidade do código, nível de experiência, revisitas e tempo de revisão), que contêm uma ligação ao estado cognitivo, já provado por estudos anteriores, o nosso objetivo é apresentar uma solução não-invasiva de baixo custo capaz de aceder à estado cognitivo do revisor, que vai permitir medir o nível de dedicação do mesmo à revisão. Os dados obtidos através dos sensores vão permitir que o revisor obtenha feedback sobre as regiões que já reviu, podendo assim caso seja necessário voltar a rever certas regiões ou não.

Nesta tese apresentamos todo o processo de desenvolvimento da ferramenta que nos permitiu fornecedor feedback ao revisor utilizado dados biométricos e não-biométricos.

# *Acknowledgements*

In first place, I would like to express my gratitude towards Professor Henrique Madeira and Haytham Hijazi, for giving me the opportunity to start this work prior to the start of the thesis, as an initialization research scholarship and also for being an excellent co-advisor, always available, giving me encouragement, motivation and help throughout the thesis. Thank you both for guiding me closely throughout the thesis, but also for the excellent guidance and all the knowledge they taught me that goes beyond the scope of the thesis. From their availability, motivation and helpful discussions to the opportunity of addressing this work, my sincerest thanks.

I would also like to thank some fellow research workers that showed up to be available for any doubt or question i had in the course of the development of the work. A special thanks to Miguel Gomes for the cooperation on the thesis, which helped me a lot with some content that was a little off my area.

Last but not least, i would like to thank my family and friends for all the support given to me throughout my whole academic course. Without them, none of this would be possible. Thank you for giving me motivation to continue and support my journey so far.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **CNS** | **C**entral **N**ervous **S**ystem |
| **ANS** | **A**utonomous **N**ervous **S**ystem |
| **SNS** | **S**ympathetic **N**ervous **S**ystem |
| **PNS** | **P**arasympathetic **N**ervous **S**ystem |
| **HR** | **H**eart **R**ate |
| **ECG** | **E**letro**c**ardio**g**ram |
| **CISUC** | **C**entre for **I**nformatics and **S**ystems of the **U**niversity of **C**oimbra |
| **BLEU** | **B**ilingual **E**valuation **U**nderstudy |
| **ROUGE** | **R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation |
| **CLT** | **C**ognitive **L**oad **T**heory |
| **LTM** | **L**ong-**T**erm **M**emory |
| **WM** | **W**orking **M**emory |
| **HRV** | **H**eart **R**ate **V**ariability |
| **ST** | **S**kin **T**emperature |
| **OSS** | **O**pen **S**ource **S**ecurity |
| **FMRI** | **F**unctional **M**agnetic **R**esonance **I**maging |
| **FNIRS** | **F**unctional **N**ear- **I**nfrared **S**pectroscopy |
| **RR** | Time elapsed between two successive R waves |
| **GSA** | **G**landular **S**weat **A**ctivity |
| **EEG** | **E**lectro**e**ncephalo**g**raphy |
| **RMSSD** | **R**oot **M**ean **S**quare of **S**uccessive **D**ifferences |
| **NN50** | The number of pairs of successive **NN** (R-R) intervals that differ by more than **50** ms. |
| **EDA** | **E**lectro **D**ermal **A**ctivity |
| **BVP** | **B**lood **V**olume **P**ressure |
| **LF** | **L**ow **F**requency |
| **HF** | **H**igh **F**requency |
| **ML** | **M**achine Learning **L**earning |
| **AI** | **A**rtificial **I**nteligence |
| **k-NN** | **k-N**earest **N**eighbors |
| **PPG** | **P**hoto**p**lethysmo**g**ram |
| **PSD** | **P**ower **S**ctrum **D**ensity |
| **NHS** | **N**ational **H**ealth **S**ystem |
| **GSR** | **G**alvanic **S**kin **R**esponse |
| **IBI** | **I**nter-**B**eat **I**nterval |
| **SDSD** | **S**tandard **D**eviation of the **S**uccessive **D**ifference |
| **PRV** | **P**ulse **R**ate **V**ariability |

*I would like to dedicated this thesis to my whole family*
*To my parents, who are the reason why i am able to achieve this level of education. Without them, none of this would be possible.*
*To my brother and sister, for their support and encouragement throughout this challenge of taking the master.*
*To my girlfriend, because she helped me keeping motivated and engaged on the work, while being an unconditional emotional support.*

# Chapter 1

# Introduction

According to the curricular plan of the Master in Informatics Engineering – Software Engineering, students had various choices to get the degree of Master (Ex: internship, project etc). In my situation, I chose to enrol on a Research Project. The choice made was focused on the Research Project since it helps juniors, like myself, to integrate and experiment the research path. This also allows to gain some knowledge about new technologies and topics inside the area of Informatics Engineering.

## 1.1 Context

This thesis work aims to develop a new tool (iReview) that evaluates the code review quality using biometric measures gathered from code reviewers (often called Biofeedback). This is done through the use of non-intrusive devices. Biometric measures such as Heart Rate Variability (HRV) and Electrodermal Activity (EDA) are used to assess the reviewer's comprehension of the code under review and the eye-tracker is used to indicate code regions that were not well reviewed. iReview evaluates the quality of each review globally and indicates the code regions that have not been well-reviewed, explaining why those code regions should be reviewed again. The tool uses Artificial Intelligence techniques to classify the code regions into good and bad reviews based on various biometric and non-biometric features.

As technology progress, automated and helpful software start to take place in the society, in order to be able to give the best help possible to the user. The "Tesla self-driving car" is an example of a software that uses a lot of technology in order to automate the process of driving, giving the driver the option to not do it. But how does Tesla car do that? The answer lies on the data that the software is constantly receiving through sensors on Tesla car.

Sensors allow us to gather data about the environment around us, which can allow us to make decisions if certain circumstances are met. Analyzing and crossing the data between various sensors allows us to make even more precise and competent decisions. For example, if Tesla self-driving considers his speed sensor and distance from the front car, he can decide when to break with more precision than he could if he would only consider the distance from the front car.Likewise, in our tool, integrating more than one sensor measures (i.e., HR and EDA) enables us to optimize the decision about the code review quality.

Code reviews have huge impact on software development, so how can we make sure that the reviews are the best they can be? What if reviewers can not make a good review due to some interference while doing the task? How can we make sure that the reviewer is giving the necessary attention to each piece of code, considering its difficulty? This thesis attempts to provide a code review evaluation tool that could answer the previous questions.

## 1.2 Motivation

Software products and platforms have become, today, omnipresent in every domain and an important asset for every company. It is estimated at [17] that in the world there are around 26.9 million software developers .

Software bugs are the monster under the bed for any tech-first business. Coders, developers, designers and QA professionals are desperate to avoid them at any cost, and when a product ships with critical bugs, it can be a hard pill to swallow. But software bugs are not just an annoyance, as many of us know, they come at a cost to fix too.

Here are statistics found in [15] and [33] that can give us a scope of how frequent bugs can be:

- On average, a developer creates 70 bugs per 1000 lines of code. World wide, (assuming every software developer makes at least 5000 lines of code per year, based on the statistics from 2020 presented in [20]) would turn out to be 134,5 trillion bugs every year world wide;
- 15 bugs per 1,000 lines of code find their way to the customers, which would make turn out to be 20,175 billion bugs reach out to the costumers world wide;
- Fixing a bug takes 30 times longer than writing a line of code;
- 75% of a developer's time is spent on debugging (1500 hours a year);
- In the US alone, nearly 100 billion euro is spent annually on identifying and fixing product defects.

But what is the real cost of software bugs? Is it really that high? Research by CISQ found that, in 2018, poor quality software cost organizations 2.8 trillion dollars in the US alone. But the expenses that software defects have, might not be only monetary. One great example of this, is the NASA Mariner 1, referenced in [1]. Official reports, issued by the Mariner 1 Post-Flight Review Board, concluded that a dropped hyphen in coded computer instructions resulted in incorrect guidance signals being sent to the spacecraft. Other sources refer to it as "overbar transcription error" and even to a misplaced decimal point.

Thus, the need to guarantee code quality arises, as a simple bug can be harmless or can also be the origin of all chaos associated with the code. Several different studies have shown the importance of code reviews and how they affect the work environment. Code reviews are not only meant for bug-detecting but also to teach and help programmers to avoid making the same bug again. And this does not have to be associated to bugs only. Code reviews are also relevant to help a programmer improve his programming skills. For example, a code reviewer might identify an optimized way of performing the same action in a code. By transmitting this knowledge to the programmer, the code reviewer is helping the programmer improve his skills.

Sadowski et al. [28] has already shown that code reviews take a very important role on code development, from detecting faults on the code, to educate and help junior developers to improve their coding experience. A good code review can detect faults and teach developers how to avoid making them again, as well as teach good code practices. This is the last barrier before a change is commited to the main code. Any defect that the new code has, will affect the main code if the defects are not identified before the change.

Roberto et al. [22] has also shown that, on average, developers spend 70% of their time performing program comprehension.

Therefore, to evaluate the code review quality, it is relevant to assess the comprehension level of code reviewers to estimate their engagement level while performing the review task. With this tool, our aim is to evaluate reviewers engagement level, and thus the quality of their work code review. While the biometric data (HRV and EDA) provides a surrogate of the cognitive load of the reviewer, the desktop low-cost eye-tracker provides a pointer to the code region with less engagement (i.e., low quality review). To the best of our knowledge, there are no tools that evaluate the quality of the code review based on personal attributes and cognitive load. Also, one of the motivations is that established litterateur [5], demonstrated the viability of using biometrics (non-intrusive biofeedback devices) to assess cognitive load, and this is compatible with the software development environment (especially with the desktop low cost eye-tracker).

## 1.3 Objectives

The main goal of this thesis is to develop a tool capable of evaluating code review quality based on the reviewer's cognitive load assessment. The tool shall indicate, while performing a review using low cost and non-intrusive biometric sensors, the regions that the reviewer did not review well so that he can give another chance for a second review. For this goal, several objectives must be completed:

- Analyze the state-of-the-art in the field of code review practices and tools;
- Define the main tool components and architecture;
- Develop the main tool for experimental setups;
- Design an experimental protocol with realistic code review scenarios;
- Collect and analyze data from volunteers described in the protocol while the reviewer performs code review tasks and equipped with non-intrusive biofeedback devices to assess their engagement level and cognitive load;
- Provide an evaluation report for the code review quality with indicators (pointers) to the code regions that were not well covered/reviewed/understood.

All the previous objectives will help us reach our goal, a tool that with data retrieved from non-intrusive sensors is able to give a report for the code review quality addressing regions that are not well reviewed.

## 1.4 Contribution

As I am dealing with a project that will reuse some components already developed in previous related works, this section will define what exactly is my contribution to the tool.

In this thesis, I introduce the following contributions:

- A computational methodology to synchronize the input signals
  Synchronization between input signals is necessary, so that we can understand where exactly the reviewer had some type of behaviour based on his HR and

EDA data. This is achievable via common timestamp. On the eye-tracker side, when a reading is performed by the eye-tracker, the timestamp of the reading is extracted based on the local computer timestamp. Each reading has also assigned a region to identify the region where the reviewer was looking at. On the HR/EDA sensor E4[1], we are given the initial timestamp of the recording, and the sampling frequency. Using both of them, we can calculate each timestamp for each reading taken.

After having both signals assigned to the respective timestamps, a comparison is made. Since the eye-tracker sampling frequency is faster, we use it as a reference. After comparing the eye-tracker timestamps with the wristband timestamps, when there is a match, between them, the first and last timestamp of the same consecutive region are taken. We then proceed to look for any HR and EDA data that is contained within that interval.

The Synchronization between the signals so that each value of the sensors we acquire match the timestamp between them in order to have a relation between the information we can deduct from those signals.

- Development of a prototype of the tool Development of a realistic tool to evaluate code review quality which will be used on experimental setups, including the design of the interface that will enable volunteers to test it.

- Protocol development
  Development of protocols and test scenarios in order to successfully acquire viable data to evaluate code review quality.

## 1.5 General outline of the thesis

This section describes the structure of this report. The remaining chapters of this report are organized as follows:

- Chapter 2 - *Background and Literature Review*, where basic concepts are explained, technical concepts will be discussed to allow a better understanding of the report, related works is analyzed and compared to our current work. It's also where is noted why this work is relevant, and the several projects that the current one can be related to and how;

- Chapter 3 - *Methods and study/experimental design issues*, where we show the architecture of the tool, functionalities and also the protocol of our experiment

- Chapter 4 - *Data Analysis*, where we analyze the data gathered in our experiment and go through feature engineering

- Chapter 5 - *Conclusion and Future Work*, where we take the broader conclusions and analyze what future work could be done to improve the tool

[1] E4 refers to a version of Empatica smartwatches

# Chapter 2

# Background and State-of-The-Art

This chapter is divided into two sections:

- Background

  In this section I give an introductory overview on the theoretical underpinnings of the tool components and the code review/comprehension.

- State of the Art

  In this section I analyze and make reference to papers and research that is related to our tool. I make a comparison and also possible relations between the papers results and how our tool could benefit from said paper.

## 2.1 Background

This work is interdisciplinary and spans Software Engineering, Biometrics and AI. Therefore, this section addresses the background of different topics.

### 2.1.1 Code Review

Code review is a software quality assurance process in which software's source code is analyzed manually by a team or by individuals using an automated code review tool (e.g., Github). Not only for finding software defects, code reviews also have other purposes, as shown in [4] and [31], code improvement, alternative solutions, knowledge transfer, team awareness, improving development process and some others.

Why is this important? Why should we spend time reviewing code that was already thought of by the code developer? When approached by a different person, a code might seem different. How different? It depends on the person reviewing it. If the reviewer is a more experienced programmer than the original developer, there may be optimized solutions that the developer is unaware of. This opens an opportunity for not only code improvement, but also for teaching. The expert reviewer can leave comments and help the developer to develop his own skills. It is also common for programmers to be so focused on a piece of code, they lose perspective of the remaining code and might not catch easy bugs, that a fresh pair of eyes can easily detect. This are just some benefits that come from code reviewing.

But Code reviewing as a task itself, is no piece of cake. It is not as easy as reading a simple text. It entails code comprehension, which by itself requires the ability to understand the language used and the logical thinking behind the developed code. So it is important to be highly engaged in the review while performing it. Thus, to

produce a high quality code review it requires high attention to the task and a good mental state (i.e. focused, stress-free).

Since the most used type of code review is tending to be Asynchronous Code Review, it also means that the Code Reviews are heavily dependent on the reviewer that is performing it. If the reviewer is not on his best mental condition (e.g. focused on the task), it is possible that the code review is not as good as it could have been if the best conditions of mental health of the reviewer would meet.

In Software Engineering has the software development industry began to rise, so did the need to ensure quality of said software, as software defects could cause big harm to the software projects and even the companies. But what damage can a bug cause you ask?

A research done by the Consortium for Information and Software Quality (CISQ) found that, in 2018, poor quality software cost organizations nearly 2.47 trillion euro in the US alone. Not only can software defects cost money, they can also put people in dangerous situations. It was stated that more than 10,000 patients in the UK were at risk of being given the wrong medication after an National Health System (NHS) glitch was discovered in 2018.

To overcome software defects, a lot of methodologies were created and throughout time, adapted to the needs of the companies. Code reviews are one of the most used and relied methodology used by big companies such as Google who also share their practices on code review in various github repositories at [14].

There are different types of code reviews. Fagan's methodology described at [8] is the most popular when it comes to Formal Code Reviews (or Code Inspections [7]) which consist on the following 6 steps: Planning, Overview, Preparation, Inspection Meeting, Rework and Follow-up. This methodology would require various members of the team/project to be present at a formal meeting to analyze code.

Today the software industry follows a lightweight version of the code review which can be done in various ways:

- Instant Code Review

  Instant code review happens during pair programming. While one developer is creating code, the other developer is reviewing the same code simultaneously, paying attention to potential issues and giving ideas for code improvement on the go.

- Synchronous Code Review

  Here the coder produces the code and asks the reviewer for a review immediately after the coding is done. The reviewer joins the coder at his desk and they look at the same screen while reviewing, discussing and improving the code together.

- Asynchronous Code Review

  This one is not done together at the same time on the same screen, but asynchronously. After the coder is finished with coding, he makes the code available for review and starts his next task. When the reviewer has time, he will review the code by himself at his desk on his own schedule, without talking to the coder in person, but writing down comments using some tools (e.g. Github, Phabricator, Gitlab etc). After the reviewer is done, the tool will notify the coder about the comments and necessary rework. The coder is going to improve the code based on the comments, again on his own schedule.

Modern Code Review is today used in various companies through asynchronous Code Review. Big companies like Microsoft, Facebook and Google use tools to assist in this way of code reviewing. Microsoft uses Github [13] and Visual Studio Team Services, Facebook have their own reviewing tool called Phabricator [21] and Google states they review each piece of code, any change to the code that is made. For this, Google uses Critique. Code reviews are so important, that big companies like the ones mentioned, even developed a tool to assist it. With tools like the ones mentioned above, we can improve the process of code reviewing, highly increasing the possibility of detecting software defects as well as maintaining the organization and a history of the whole process of code reviewing. Although there are widely available code review tools, there are no tools, up to our knowledge, that evaluate the code review quality based on human factors, which we present in this thesis.

Code review is also a high mental effort task. Siegmund et al. [29] shows that for comprehending source code, five different brain regions become activated, which are associated with working memory, attention, and language processing — all fit well to our understanding of program comprehension.

A pertinent question surges as code reviews are more and more relied on single reviewers (and as consequence, dependent on their mental effort while performing the task): How can we make sure the code review was done properly? How do we know if the reviewer had the mental conditions to achieve a desired quality for the code review? Did the reviewer fully comprehended the code he was reading?

Here is where our tool could provide a solution by using biofeedback proven to capture the manifestation of the ANS induced by mental tasks like the code review. Biofeedback is able to provide feedback about the engagement of the reviewer in each region of the reviewed code. Taking also non-biometric features in consideration, (i.e. code complexity, experience level, programming language, among others) it can adapt to the reviewers experience and enhance his feedback in order to retrieve the best outcome from the feedback provided for each region.

### 2.1.2 Code Comprehension

Code comprehension is a mandatory step in code review. Code comprehension refers to the ability of understand code. Different activities stimulate different regions of the brain. The act of code comprehension uses logical thinking, problem-solving, and thinking outside of the box. It also includes natural language understanding, as it concerns the syntax of the used language.

This task can be very time consuming, as each programmer has its own way of programming. Usually, and the most known practice to help code comprehension be less time consuming, is to document the code for other programmers to read and help to understand the code. Unfortunately the shortage of well documented code requires the need of other solutions to overcome this issue. Thus, a lot of research on understanding how can this process be optimized, exist.

The exercise of the respective responsible areas of the brain for the different types of comprehension (logical, problem-solving, natural language and out of the box thinking) provokes changes on the Autonomous Nervous System (ANS), that trigger certain responses when faced with different scenarios. Said triggers can then be read by the biometric devices, which are analyzed and can be interpreted. There are several features and different evaluation mechanisms that allow to retrieve the classification of code comprehension [11], [34],which also show us that lightweight biometric sensors can be used to accurately recognize comprehension.

The previous question still persists: How can we make sure that the code comprehension was done correctly? To answer this question, we must first understand what Cognitive load is.

### 2.1.3 Cognitive Load

Code review and code comprehension and other mentally demanding tasks induce higher cognitive load in humans. Cognitive Load is defined as the amount of information that working memory can hold at one time. It is pretty much like RAM in a computer, only for the human brain. When memory usage on your computer is high, the system starts to have issues keeping up with everything that is going on. The same principle applies when cognitive load is high, and this load is not properly dedicated to the right activities. This could end up result in lower quality work.

Cognitive Load Theory (CLT) is an instructional theory that reflects the way we process information [18]. It builds upon the widely accepted model of human information processing [3]. It describes the process as having three main parts: sensory memory, working memory and long-term memory. The end goal is to pass information from the working memory to the long-term memory, so that the learner can acquire the information and be able to remember it when he needs it. Not all information is necessary, and the presence of distractions creates unwanted information, which might makes the end goal harder to achieve.

The working memory is responsible for rapid perceptual and linguistic processing. it works out what the new information is all about and whether to store it in the long-term memory or discard it. Sweller et al. [32] stated that, since working memory has a limited capacity, instructional methods should avoid overloading it with additional activities that do not directly contribute to learning.

Cognitive load theory differentiates the types of cognitive load into 3 types

- Intrinsic load

  Intrinsic cognitive load is the inherent difficulty of a topic. It can not be changed. For example, understanding basic math operations (i.e. addition, subtraction) is way easier than it is to understand algebra or calculus. There is no way the difficult of the matter can be changed. What can be changed, is the way the matter is taught, which might make a heaver intrinsic load activity, easier to understand. Shifting to the context of our work, when a reviewer is faced with a code, the difficulty of the code can not be changed. What can change, is the reviewer experience on understanding the code. If the code has a good concise documentation, it will make it easier to comprehend.

- Extraneous load

  Extraneous cognitive load does not aid in the learning task. When facing a new subject or topic, everything that is not helping in the learning process, is considered extraneous load. Distractions such as a noisy environment, contribute towards extraneous cognitive load. In our case, anything that influences the reviewer to get distracted or to reduce his attention towards the code he is reviewing, can be considered extraneous load.

- Germane load

  Germane cognitive load is the desired result of comprehension. It is the capacity of the working memory to link new ideas with information in the long-term memory. It can be identified as the moment when someone understands

> the logic behind an idea. In our context, it can be described as the moment a reviewer understands the logic behind a function, class or any component related to the code he is reviewing.

In a tech oriented approach, broadly speaking, we want to minimize the Intrinsic load (through training, pair programming, choosing good technologies), eliminate Extraneous load (i.e. reduce superfluous tasks, improve workspace) so that we can have more space for germane cognitive load, which is where we retain and understand information (new technologies, concepts, code).

From a Software Engineering perspective, cognitive load is a measure of the amount of effort being used in the working memory. Different tasks will affect reviewers differently showing different measures of cognitive load. Higher mental demanding tasks, that involve different abilities, like reviewing a code which involves logical thinking and language understanding for example, are way heavier on the mental load then looking at a blank screen or reading a natural language text. The latter would be expected to have lighter cognitive load and present less mental stress.

Our goal with this tool is to assess and evaluate the engagement of a reviewer during his review and alert the review for regions which might have been not properly reviewed. Implementing it in IDEs would make them cognitive-aware.

### 2.1.4 Biometrics

To understand how we can gather information about a reviewer's cognitive state, we must first understand the ANS, the system that makes the human being have different reactions to different stimuli [2].

The autonomic nervous system is a component of the peripheral nervous system that regulates involuntary physiologic processes including heart rate, blood pressure, respiration, digestion, and sexual arousal. It contains three anatomically distinct divisions: sympathetic, parasympathetic, and enteric.

The Sympathetic Nervous System (SNS) and the Parasympathetic Nervous System (PNS) contain both afferent and efferent fibers that provide sensory input and motor output, respectively, to the Central Nervous System (CNS).

Activation of the SNS leads to a state of overall elevated activity and attention: the "fight or flight" response. In this process, blood pressure and heart rate increase, glycogenolysis ensues, gastrointestinal peristalsis ceases, etc. The SNS innervates nearly every living tissue in the body. The PNS promotes the "rest and digest" processes; heart rate and blood pressure lower, gastrointestinal peristalsis/digestion restarts, etc.

Sensors allow us to gather data while the reviewer is reacting to the stimuli. In sum, it allows us to read the deviations that are made in the reviewers ANS . With this, we are able to determine the cognitive state of the reviewer, which can allow us to make decisions if certain circumstances are met. Analyzing and crossing the data between various sensors allows us to make even more precise and competent decisions regarding the necessary cognitive state that is considered to be needed to perform certain tasks.

This then allows the extraction of signals such as Heart Rate (HR), Skin Temperature (ST), eye gaze, among others that, when crossed together, allow to make certain conclusions. Previous research has already proven that certain features are related to the mental state and can be related used to measure Cognitive Load [9]. Research

as also showed that it is indeed possible to use biometrics to predict code quality [23],[11].

As the reviewer is performing the review, he might experience different mental states over the course of the review. These states will reflect in the signals gathered, such as HRV and EDA and eye-gaze. Considering the eye-gaze for example, a reviewer that is not understanding well a method or function, will most likely re-read the whole method from the start to try and pick up the logic of it. Thus, it is expected in the eye-gazing data to see something like a "loop" in the eye-gazing coordinates (x and y), as the reviewer keeps re-reading the code until he understands. If we also consider HRV and EDA in this scenario, we would expect the user to feel some discomfort. It is hard to tell exactly what would happen, as each individual reacts differently to mentally challenging scenarios, but it would be expected for the user to feel stressed, which can be identified by the LF/HF ratio feature extracted from HRV. This feature shows the balance between the SNS and the PNS, which could tell us how mentally challenged the reviewer might be feeling based on his LF/HF ratio. A high LF/HF value would mean that the reviewer has encountered a difficult topic and is mentally demanding.

As our tool aims to be non-intrusive and low cost, with the technology available nowadays and the development of wearable devices industry, it is easy to find devices that allow us to monitor heart rate variations but are still non-intrusive and low cost. PPG is a technology that uses a light source and a photo-detector at the surface of skin to measure the volumetric variations of blood circulation, which allows for heart rate monitoring in the wrist, usually acquired through wearables such as smartwatches and fitbit wristbands. Gil et al. [12] studies the correlation between PPG and HR, concluding that Pulse Rate Variability (PRV) (the estimation of variation in heart rate from a Photoplethysmogram (PPG) signal) and HRV have no meaningful differences that forbid the use of PRV as a surrogate of HRV. Pinheiro et al. [25] also concludes in his study that most PRV indexes may be used as surrogates to HRV.

Some of the wearables today also include EDA sensors at an affordable price. Usually people already own this type of devices and are already used to wear them on a day to day basis. Most fitbits and smartwatches have PPG readings with some of them also having EDA. Some of them are used for exercising, others for a casual day to day use, but these same devices could also be used on a development environment. We can surely find people already wearing their smartwatches or fitbit while performing their work.

## 2.2 State of the Art

As we know, code reviews are a great way to prevent bugs that were left behind by a developer, to reach the deployment stage. Not only that, code reviews can also be a way for junior developers to learn how to code better and recognize the mistakes made so that they can improve their coding skills. Thus it is important to guarantee that code reviews are well done.

Sadowski et al. [28] explain how Google perceive Code reviews, why do they use it, methods that were/are used to follow such reviews, motivations and other aspects of Google regarding code review. Google performs code reviews on every piece of code they make! Their approach is mostly the asynchronous approach, where the reviewer reviews the code that is sent to him, takes some comments and gives some feedback regarding the same code. Google also uses GitHub in order to maintain an organization or their development process, and Critique as a reviewing tool. But how can Google ensure the reviewer is ready and able to perform the task he is assigned to? What variables need Google to consider when giving a task like this?

Here is where our tool shines and could come to improve the process of code reviewing. Our tool aims to ensure that the review was given with the right attention and comprehension. This could improve code reviews, helping the reviewer to find more bugs, improving not only the quality of the code, but also the quality of the education of the developer, given that our tool provides a new way of feedback reports on the code review quality based on biofeedback, which helps evaluating the engagement and understanding of the code under review. He could then re-review the regions he had a lesser engagement on and understand that region better, resulting on a better overall report on the quality of the code review. Understanding the region itself will also enable to the reviewer to understand the logic behind it and present alternative solutions or improvements. In case the region was not fully understood, the reviewer could not have the knowledge to suggest improvements.

Various studies have addressed biometrics in code comprehension and review as a basic construct of measurement. Initial studies in the field started by analyzing the impact of code review on the brain using Functional Magnetic Resonance Imaging (FMRI). Results from those studies motivated researchers to look into less intrusive alternatives, as the heavy sensors were unsustainable due to their cost and physical dimensions.

Fucci et al. [11], based on a previous paper that used fRMI to understand the weight of code reviewing on the brain activity, tries to understand if the use of lightweight biometric sensors to measure human physiology is supported by the results. The authors use two measuring devices - a health tracking wristband Empatica E4 and BrainLink EEG headset. The experiment consists on the comprehension of code and prose. A total of 28 volunteers participated (24 male and 4 female), with different ranges of code expertise. During the experiment, the objective was to be able to understand which task a participant is undertaking based on signals collected from lightweight biometric sensors. Results show that HRV signals can be used to distinguish what task the volunteer was performing with high accuracy.

This prove to us that it was possible to replicate a previous study that used heavy sensors (e.g. FMRI) replacing them with lightweight sensors.

As technology develops, there are now sensors able to collect biosignals at lower prices. Not only that, some of the sensors are already familiar to the common user (e.g. smartwatches and fitbits). Thus, researchers started to consider the possibility of taking advantage of those sensors to gather biometrics in a software engineering environment.

Couceiro et al. [6] present emergent experimental results showing that mental effort of programmers in code understanding tasks can be monitored through HRV (heart rate variability) using non-intrusive wearable devices. Their goal was to investigate how mental effort in reading and understanding programs of different complexity can be measured by a set of sensors placed in the programmers (volunteers) that participated in the experiment. The experiments in [27] consisted of the analysis of3 code snippets in Java. In their experiments they used EEG, ECG, EDA and eye-tracker sensors.

Important to note that the HRV was gathered from the ECG which is not the best solution for a real environment. Instead, using wearables (such as smartwatches or fitbits) to replace the ECG sensor would fit the real environment. The Pulse Rate Variability (PPR) analysis would be used as a surrogate for HRV.

But would the PPR be suitable as a surrogate for HRV? Can they be reliable enough to replace ECG sensors?

Pinheiro et al. [26] experiment and aim to use PPG as a surrogate toHRV. Because HRV needs the ECG signal, the use of ECG sensors is needed. Nowadays, and with the increasing improvement on technology, devices like smartwatches and fit bands are quite common. These devices are able to give heart related feedback. Instead of the HRV,the watches/fit-bits extract a PPG signal. The authors analyze PPG features as surrogates for HRV indexes, in three different contexts: healthy subjects at rest, healthy subjects after physical exercise and subjects with cardiovascular diseases (CVD). Results confirm that the majority of PRV indexes may be used as surrogates for ECG-based HRV in healthy subjects at rest.

This paper is really important, as it states that with the use of sensors that use PPG, which are usually non-intrusive and affordable, (usually, people use wear them on a daily basis) might be possible to extract biometric features capable of replacing HRV. This study supports our tool, as our goal is to use non-intrusive low cost sensors and smartwatches/fit-bracelets are one possible option.

For example, Fritz et al. [24] used biometrics, such as HRV and EDA, to automatically identify code quality concerns while a developer is making a change to the code. The aim is to prevent code defects online lowering the development cost. The authors recruited 10 volunteers, working on the same project. Their focus was to obtain biometric features (for all ten study participants, they collected skin temperature, HR, HRV, RR data and for some participants EDA as well) but also considering other metrics such as code metrics, interaction metrics and change metrics. For the classification they used Weka, a Java-based Machine Learning framework. For their study they used two sensors, Empatica E4 wristband and SenseCore chest strap. The results of their study suggested that developers biometrics can indeed be used to determine the perceived difficulty of code elements and furthermore to identify places in the code that end up with code quality concerns, such as bugs. Although the SenseCore chest strap is not the ideal solution for a real work environment, the E4 wristband used in the experiment would be a good choice as it is a non-intrusive wearable, similar to a smartwatch/fitbit.

Fritz et al. [10] demonstrate the potential that biometric data can have to accurately and instantaneously measure perceived task difficulty, progress and incorruptibility of a developer. Their results support the fact that biometric data has the ability to reveal information regarding the developer's cognitive and emotional state. This offers much promise to provide better developer support and improve individual productivity as well. At the same time there are still several challenges to overcome for this to become a reality and widely accepted by developers, such as privacy concerns or sensor limitations. The aforementioned paper agrees with our assumption that biometrics, and particularly wearable devices can indeed capture the cognitive and emotional states, which is what we are measuring through specific features to give feedback to the reviewer.

Hijazi et al. [16] made an early evaluation of our tool that evaluates the code review quality using biometric measures gathered from code reviewers. A controlled experiment was designed to examine the viability of the tool concept. In the experiment, the volunteers were equipped with cardiac sensors to ensure reliable and precise measurement of the heart signal for the evaluation purposes .The participants were given a set of programs to review while equipped with non-intrusive sensors and an eye-tracker. The programs had injected bugs unknown to the participants so that their behaviour whether by finding bug or missing it could be recorded, using features such as the Low Frequency (LF) over High Frequency (HF), highly sensitive to the sympathetic and para-sympathetic nervous system changes. They then classify each code region as either well or badly reviewed using AI mechanisms and ML.

Performing these kind of protocols, controlled experiments in controlled environments in order to minimize the "noise" in the data, help us to better understand the behaviour of programmers before a bug is detected or is missed. By recognizing such patterns we are able to identify and provide correct feedback when those situations happen. Experiments like these, help our tool development, as we aim to use similar protocols in order to achieve and give the correct feedback at the most appropriate time window. Although several studies addressed code review, code comprehension, and even assessing the quality concerns of code review using biometric features, this thesis introduces a drastically new intelligent biofeedback tool that evaluates the code review quality at granularity level of code region while indicating the code regions that were not well reviewed. Using already known gathering techniques of biometric features and signals, proven by research to have an impact on the cognitive load of the reviewer, our tool uses such biometric features to be able to provide feedback to the reviewer.

# Chapter 3

# Methods and study/experiment design issues

This chapter covers the tool architecture, design and main functionalities that were implemented in the tool. Then we go through the experimental design and protocol of the data acquisition process to evaluate the tool.

## 3.1 iReview - Preliminary version

iReview aims at evaluating the quality of the code review using biometrics measures that assess the mental workload while doing the code review. In the current implementation of the tool, the mental workload is assessed through HRV extracted from the Blood Volume Pressure (BVP) signal and EDA gathered by E4 smartwatch. There are currently many alternative non-intrusive devices to measure HRV such as many smartwatches or wrist bracelet fitness devices equipped with HR and EDA sensors. We chose Empatica since it showed higher precision and more praticality in measuring EDA.


FIGURE 3.1: iReview Physical Setup

In Figure 3.1 we can visualize the physical setup required to use the tool. The setup is composed by the following devices:

- Low-cost eye-tracker

  The eye tracker used (Tobii 5L) was designed for the video game market and because of that is quite affordable (it costs around 260 euro). It gathers pupillometry information and gaze information. In our tool, we mainly use this device to track the gaze coordinates (x and y), which indicate where the reviewer is looking.

- Non-intrusive wearable device

  The wearable used in our tool is the Empatica E4 which contains HR and EDA sensors capable of recording those signals. This wearable, like many other smartwatches/fit bracelets, are non-intrusive, usually worn on the wrist and are mainly the source of measuring the cognitive load of the code reviewers while they are performing the review task.

One aspect that should be emphasized in Figure 3.1 is the fact that the sensors used for the biofeedback are not intrusive at all. The Tobii eye-tracker is at the bottom of the screen and the HRV can be measured through a smartwatch or an HRV bracelet device.

To better understand how our tool will work, the following diagram shows the main components of the tool.

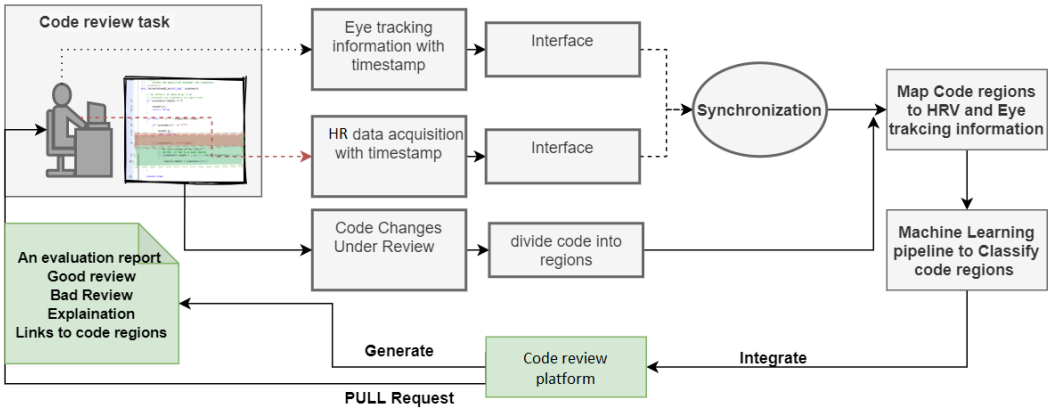


FIGURE 3.2: Schematic Diagram of the tool

Figure 3.2 shows the tool schematic design and the expected outcome. These were the first thoughts regarding the tool functionalities and expected results. To better approach the diagram, we will split it into the following components:

- Eye-tracking information with timestamp

  Using the eye-tracker Tobii 5L, we are able to access the data being recorded from the eye-tracker while the review is being performed so that we can know where the reviewer is looking. While the device is measuring, we also take a local timestamp whenever a measure of the data is recorded. This way, each measure will have an associated timestamp that can be used to perform the synchronization between signals used.

- HR and EDA data acquisition with timestamp

  Using E4 Empatica wristband, we are able to extract HR and EDA signals while a review is being performed. This sensor already includes a starting timestamp

of the recording. It also gives the user the sampling frequency, so that it is possible to calculate each measure timestamp based on the starting timestamp of the review. This timestamp and each timestamp calculation for each measure, will be used to perform the synchronization between the signals of the Eye-tracking and HR and EDA data.

- Synchronization

  The synchronization between signals is done via common timestamp. Using the fastest sensor as a reference (the eye-tracker), for each measure taken we look for a correspondent measure taken from the HR and EDA sensors. This way, we can associate HR and EDA data to a specific region registered from the eye-tracker. We then split the regions into the respective regions so that we can analyze the data on a code region granularity level. The synchronization is performed in MATLAB due to the availability of several toolboxes, commonly used in the development of signal processing algorithms, such as the Signal Processing Toolbox and the Statistics and Machine Learning Toolbox.

- Mapping code regions

  Mapping the code regions of the code under review to the HRV and EDA data and eye-tracking through the common timestamp. The tool previously performs the division of the code under review into code regions. The goal is to perform the biofeedback analysis of the reviewer's performance focusing on smaller portions of code. These code regions consist of non-overlapping code chunks that represent a coherent sequence of code lines, following as much as possible the natural way the reviewer reads the code. For example, a sequence of assignments and computations is a region, a repetition structure containing a cycle (maybe with other cycles inside the cycle) is another region, and so on.

- Machine Learning pipeline to Classify code regions

  This component will fetch the output from the synchronization, and evaluate all the data at a code region granularity. It will then provide a report on the interface it is implemented in with the a classification of the review (i.e. good or bad), an explanation for the classification and a link to the code regions that were considered in the evaluation.

  The machine learning pipeline consists of features extraction, feature selection, fitting, and prediction, hyperparameter tuning, cross-validation, and testing. In the feature extraction module, the main objective is to pre-process the collected HR signal and segment it so that the extract RR intervals are fed into the algorithms that will extract the variability of the heart rate (HRV) indexes (in time and frequency domain HRV features). After the features are extracted, the best (and least redundant) features are selected. These will represent the inputs for the classification model and are used, not only to train and validate the proposed model but also to tune its parameters.

  The model for the classification of the quality of the code review in each code region uses an interpretable machine learning technique such as K-nearest Neighbors, Decision Trees, and other classifiers to make it possible to explain

the reasons why code regions have been classified as being bad reviewed.

In short, the information used by the classifier to classify the quality of the review at the code region level, as well as to interpret/explain the reasons why a code region is classified as "bad" review, includes the following elements:

- Reviewer's cognitive load provided by HRV and EDA;
- Reading time of the code region;
- Complexity of the code region provided by Vg;
- The number of revisits to the code region;
- Experience of the reviewer.

Currently, iReview assumes a simple classification of reviewers' expertise in two levels: non-expert and expert.

Finally, the validation of the proposed tool is performed using a cross-validation scheme to avoid biased results and over-fitting.

### 3.1.1 Functionalities

For the first interface design, in order to be able to perform test scenarios and test the tool, we proceeded with a basic interface of the tool, that would be simple and intuitive, avoiding causing the volunteers any stress from not knowing the interface or it being too complex.


FIGURE 3.3: iReview Interface

Figure 3.3 shows iReview Interface that can be split into 3 simple areas:

- The button area where the reviewer can interact with the review:
  - Start
    This button will begin the review if a code to be reviewed is already selected. By pressing start, the sensors will start the recording of data;
  - Pause
    This button pauses the review and the recording of data;
  - Resume
    This button will resume the review and the sensors will start to record the data again;
  - Stop
    This button will finish the review and the sensors will stop the recording, storing the data in txt files;
  - Select a file
    This button lets the user choose a code that will be reviewed. The code will be presented in the area beneath the button taskbar (in white background).
- The text area where the selected file to be reviewed will be shown to the user when the review starts;
- The annotator is a software plug-in that enables the code reviewer leave comments for each code region/line. These comments are typical and essential in code reviews.

The code review evaluation starts when the reviewer selects the codes he wants to review and clicks the Start button. If there is no code selected, the tool will not allow the start of the review. The code can be selected through the "Click to choose a txt file" button that only accepts txt based formats. As you press start, the interface requires confirmation that the wearable device (Empatica E4) is recording. This is achieved by making one press of approximately 3 seconds on the wearable device button. After this the device starts the acquisition of HR and EDA data. After confirmation, the eye-tracker start recording eye movements. The review can be paused and resumed explicitly in the corresponding buttons of the interface. By clicking the Pause button, the eye-tracker ceases recording pausing the review. The reviewer can then resume the review by clicking the Resume button which will start the eye movements recording again. Finally, the review can stop the review using the button End which will stop the recording of eye movements. The wearable device also needs to be stopped after the End button is pressed, repeating the same press previously done. After concluding the review, 1 file is generated containing the eye movements recorded. The HR and EDA data are available to download at the Empatica website.

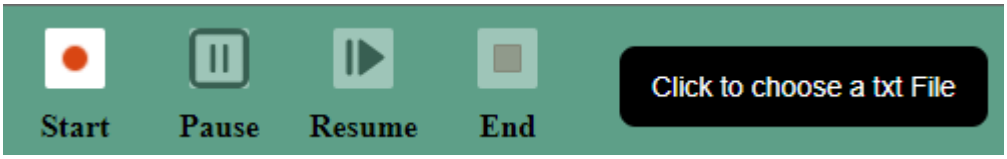


FIGURE 3.4: iReview buttons

Figure 3.4 shows the buttons available in the interface.

After setting up the main button interface, I implemented an open-source Annotation System. This open-source Annotation System will allow the reviewer to make comments or highlight code. This is an important aspect of the tool as it aims to recreate the realistic way code reviews are performed. They are important since it is through them that reviewers can highlight mistakes or comment on code that can be improved or they have doubts about. It is an essential functionality to a reviewing tool. Thus the necessity of including an Annotation System.

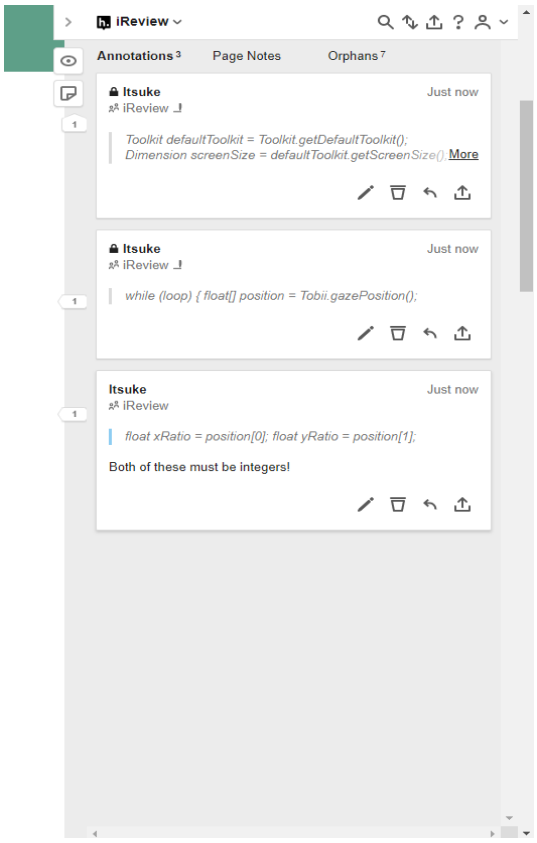


FIGURE 3.5: Annotator

Figure 3.5 shows a closer look on the Annotation System.

This Annotator was chosen because it is user based. A user can create a group and all the invited members to the group will have access to the notes or highlights that the reviewer did. This is useful for our experiment, providing us access to the volunteers comments and highlights.

After the review is concluded and the data gathered, we extract the real performance of the volunteers (Bugs found, bugs not found and non-bugs found) and proceed with the synchronization of the eye movements and HR and EDA data. From the synchronized data we then extract HRV and EDA features. Along with the real performance of the volunteers, we use the features in our Machine Learning pipeline which will provide the following:

- Global evaluation of the code review as well or badly reviewed.
- Identification of the code regions that have not been well-reviewed and might need a second look from the reviewer.
- Explanation of the Machine Learning outcomes through implementing explainable models such as K-NN which is intrinsically explainable.

## 3.2 Experimental design and Protocol

This section tackles the necessary steps needed to perform an experiment aiming to evaluate the tool

### 3.2.1 Volunteer Recruiting

Firstly the group of volunteers were introduced a data gathering consent which all approved and agreed with. After the consent, the group of volunteers were evaluated through a screening test. This evaluation consisted on understanding the expertise of the volunteers in C programming language, so that they could later be categorized into two levels of expertise: Expert and Non-expert. For our experiment, 5 volunteers were selected. The 5 volunteers are all male and consisted mainly in fellow researchers from CISUC. Through the screening test, the volunteers were asked how many lines of code had they program in C in previous years, experience in the language and how many code reviews did they perform in the language. To be expert, the volunteers need to have over 5 years of experience. over 1000 lines of code written in C and performed over 10 code reviews.

Before the start of the experiment, one of the mandatory steps is to perform the calibration of the eye-tracker. Each volunteer was asked to do this before he beginning of the experiment

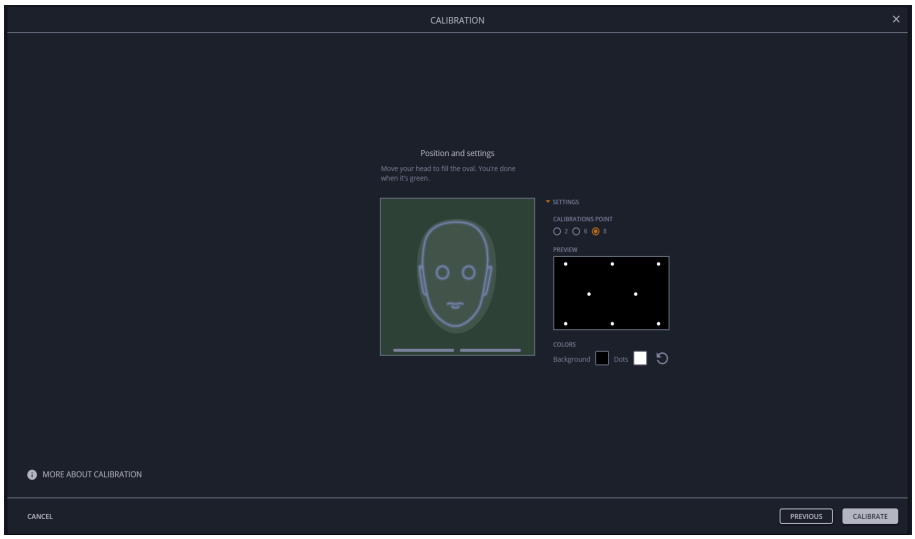


FIGURE 3.6: Calibration screen

In Figure 3.6 we can see the calibration window. The calibration screen has the positioning area and the calibration process. The positioning screen shows how the head of the volunteer is positioned from the point of view of the the eye-tracker. The volunteer should position himself so that his head stays inside the oval area suggested by the calibration software. The background will turn green when the head is in a suitable position.

On the right side of the positioning screen we have the calibration area. The calibration area allows to choose how many points will be used for the calibration. In order to achieve the best accuracy, we choose 8 dots. The dots are located in specific spots as shown in the figure. Each of the dot has x and y coordinates (defining their position on the screen). After setting up the positioning and calibration options, you proceed to the calibration itself. The configuration software will then show one white dot on a black screen.

Figure 3.7 shows the example of 1 dot being shown in a black screen. By looking at the dot accurately, it will explode and another white dot will appear until the number of selected dots are exploded successfully.

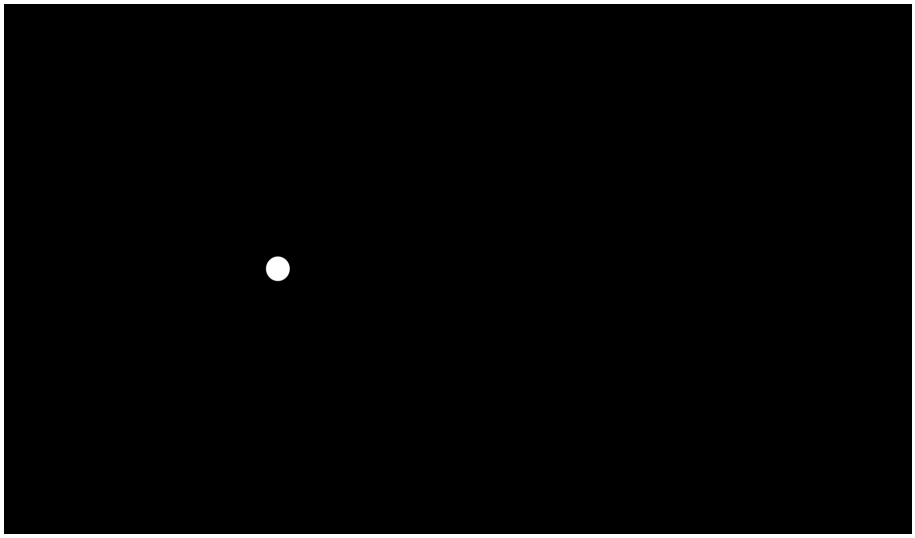

FIGURE 3.7: Example of calibration dot

At the end, it will be shown how accurately the eyes were looking at the dots.

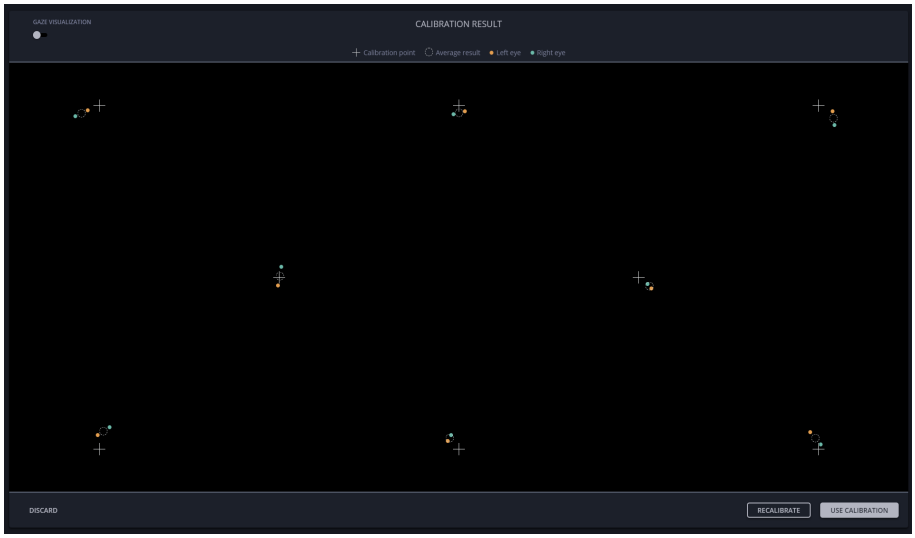


FIGURE 3.8: Example of calibration result after the calibration is performed

Figure 3.8 shows the outcome of the calibration after all the dots are successfully exploded. If satisfied with the outcome, the volunteer may proceed to start reviewing. If not, the volunteer may retake the calibration test. If the volunteer chooses to proceed, the performed calibration will be used in the eye tracker.

### 3.2.2 Protocol Description

During the experiment, volunteers had several tasks to perform before and after code reviewing. Table 3.1 shows a brief description of every task performed and how much time it took for volunteers to perform it. The tasks are displayed by the order they were taken during the experiment.

Volunteers were provided with three different code snippets in C presented in a random order to avoid biased data. Each snippet had different complexity levels.

TABLE 3.1: Table of tasks in the protocol

| | Description | Time Spent |
|---|---|---|
| Consent | Reading the data gathering consent | 2 to 3 minutes |
| Screening | Evaluation of the volunteers through a screening test to understand the expertise of the volunteers in C | 5 to 10 minutes |
| Calibration | Calibration of the eye-tracker | 1 to 2 minutes |
| Code review | Code review of the C code snippet (This step is repeated 3 times) | 1 to 9 minutes |
| Questionnaire | NASA-TLX Questionnaire after the code review (This step is repeated 3 times) | 2 to 3 minutes |

They are categorized into three levels: simple, medium, hard. Before each code snippet, a grey screen with a black cross on the center of the screen was presented to the volunteers. The volunteers were advised to think of nothing and try to abstain from any distraction, in order to achieve a rested mental state, to later be used as a baseline for the data analysis. The grey screen was present to the volunteers during 30 seconds. The participants were also asked previously, to highlight code bugs during the code review.

After each trial the volunteers were asked to fill a survey based on NASA-TLX (Task-Load Index). The NASA-TLX is a tool for measuring and conducting a subjective mental workload assessment. This allows you to determine the mental workload of a participant while they are performing a task. Our survey consisted on 5 questions ranging from 1 to 6 in order to assess the volunteer mental demand, temporal demand, performance, effort and frustration.

### 3.2.3 Data Acquisition Setup

For data acquisition, the experimental setup comprised two sensors. One wristband capable of reading Heart Rate (HR) and Electro Dermal Activity (EDA) signals and an Eye-tracker capable of reading eye movements. We used HR and EDA because they showed high capacity of indexing cognitive state (mental and other emotion states) while performing intellectual tasks. The HR signals were acquired using the E4 wristband, from Empatica with a default sampling frequency of 1Hz. The EDA signals were acquired using also the E4 wristband, from Empatica with a default sampling rate of 4Hz. The Eye-tracking device used was the 5L from Tobii with a default sampling rate of 33Hz.

In Figure 3.9 we can see the Empatica E4 wristband sensor worn on the wrist. We can also notice the Eye-tracker (with infrared sensors) connected via USB to a USB Hub that has also connected to itself, a USB Dongle.

HR and EDA signals can be recorded in 3 different ways:

- link via bluetooth to a smartphone, enabling you to get a live feedback of the recorded data;
- pushing the button for approximately 3 seconds;
- connected to a streaming server provided by the manufacture to continuously have direct access to the sensors data. This setup also requires a specific USB Dongle compatible with the streaming server software.

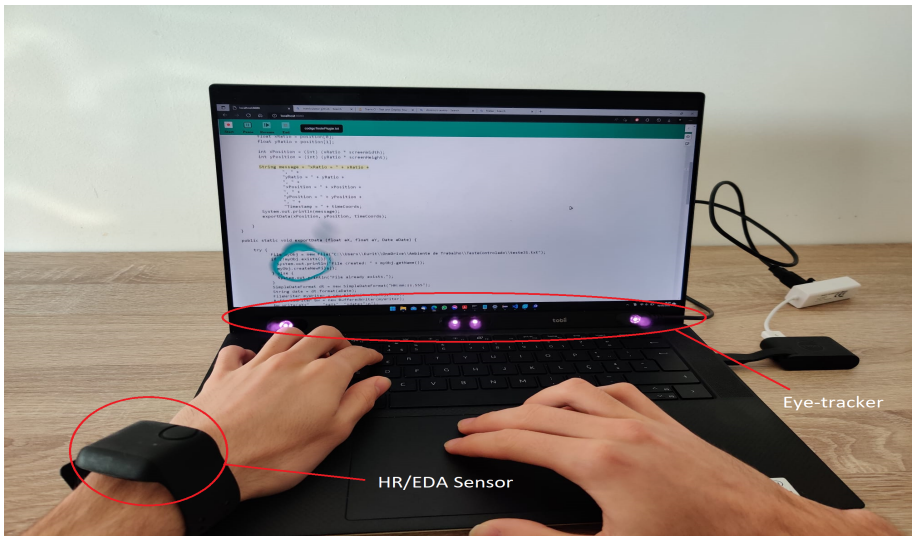


FIGURE 3.9: iReview setup

We chose to record the data by connecting the smartphone via bluetooth to the E4 smartwatch, to ensure the data gathering was performing correctly by monitoring the the live feedback in the smartphone.

To record the Eye-tracking signals, in the data acquisition setup designed, the 5L is connected directly to the computer via USB.

# Chapter 4

# Data Analysis

In this section we will start with feature engineering and end with a global analysis.

## 4.1 Feature Engineering

After gathering the data from the reviews, the next step is to extract relevant features concerning the signals gathered and also complexity metrics from software testing. We cast features from the domain knowledge and new features correlated with cognitive load.

- HR and EDA sensor

  From the gathered signals through the HR and EDA sensor we can extract relevant features from the domain knowledge, such as Low Frequency over the High Frequency, which is a feature extracted from the HR signals that shows the behaviour of the sympathetic and parasympathetic nervous systems. These two nervous systems work together to keep a balance. By measuring fluctuations of those systems, we can acquire information about the cognitive state of a reviewer. Low Frequency over High Frequency is able to show the behaviour of both systems and how is the balance between them. Any triggers on both nervous systems can be seen by analyzing this feature.

  We also extract features out of the domain knowledge correlated with Cognitive Load. Root Mean Square of Successive Differences (RMSSD) is calculated by taking the square root of the mean of the sum of the squares of differences between adjacent NN intervals. This feature is measured in milliseconds and the higher the mental load during the task, the shorter the interval is. Similarly, Standard Deviation of all NN intervals (SDNN) that is measured by taking the standard deviation of all NN intervals, Standard Deviation of the Successive Difference (SDSD) measured by taking the standard deviations between successive differences and S1/S2 which are usually shown in a Pointcaré plot (that shows the correlation between two sets of data) all have present shorter intervals when the reviewers exhibit higher mental load.

  The EDA also provide us with meaningful features since the EDA reflects the activity of the sympathetic nerve on sweat glands. By extracting the peaks of EDA, we could identify a moment of stress since the EDA is related with the sympathetic system, that responds under stressful situations. This would help to identify and confirm with other features which code region was more stressful. EDA also provide other features that result from sympathetic neuronal activity. It includes both tonic Skin Conductance Level (SCL) that generates a constantly moving baseline (which relates to the slower acting components

and background characteristics of the signal the overall level, slow climbing, slow declinations over time) and phasic components that are thought to reflect general changes in autonomic arousalSkin Conductance Responses (SCR).

- Eye-tracker

  From the eye-tracker, in order to keep track of where the reviewer is looking at, we extract the gaze position, specifically the X and Y gaze coordinates. By having the gaze coordinates of the eye-tracker, we can know where the reviewer was looking at a given time. Each measure of the eye-tracker has a timestamp associated to it.

  By analyzing the gaze coordinates or eye movements, we can understand the path the reviewer took while reviewing the code. The behaviour of the eye movement can give us important information about how he is perceiving a piece of code.

- Cyclomatic Complexity

  Cyclomatic Complexity in Software Testing is a testing metric used for measuring the complexity of a software program. It is a quantitative measure of independent paths in the source code of a software program. Cyclomatic complexity can be calculated by using control flow graphs or with respect to functions, modules, methods or classes within a software program. For the calculation of these metrics we use CCCC metrics - C and C++ Code Counter.

- Reviewer behaviour features

  Aside from biometric features, we also consider non-biometric features such as time spent on a code region and revisits. These two features are important as they help to understand the behaviour of the volunteer during the review. The number of revisits are extracted by the eye-tracker when a volunteer looks to a new region for more than 1 second. We choose 1 second, since this was the minimum time for the smartwatch to generate both EDA and HR readings.

## 4.2 Global Analysis

In our experiment, we have gathered several features from both EDA and HR signals. EDA has a feature that represents the moment when the reviewer is experiencing stress. SCR shows the peaks of EDA, which is triggered when the reviewer feels stressed. Unfortunately, due to the few number of volunteers, I could not receive significant amount of pure signals of the SCR.

Spending time in the code region might be a good indicator of the review quality. For example, as we can see in Figure 4.1 one of the non-expert volunteers, had the most time spent reviewing the codes. When compared to the other volunteers, one interesting fact is that, he performed as well as the expert volunteers.

The mean of the review time (i.e., spent time in a code region) is 30% higher in non-expert compared to expert reviewers. However, we have a case of non-expert reviewer who performed very well in detecting the bugs while spending the longest time in all codes. This means that for beginner reviewers, spending more time in a given code might give a good quality indicator.

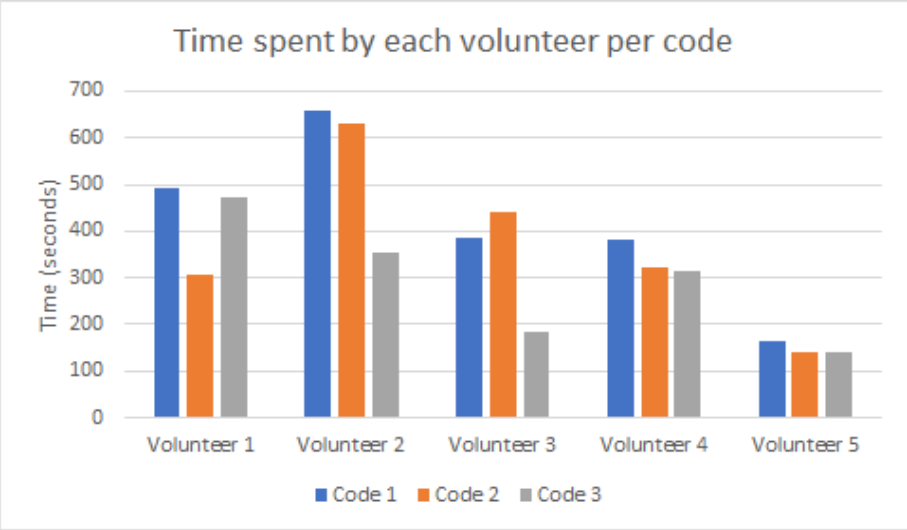


FIGURE 4.1: Caption

However, spending too much time on a region might indicate the opposite. Volunteer 1, non-expert, spent two times more than the average on a region of code 1. He highlight 4 pieces of code as bugs on a region that contained zero bugs (False Positives). This might indicate that he did not understood the code correctly or the logic behind the region we spent most time on. This might also cause the individual to have a over stress which can lower the quality of the review.

Thus, spending time in the code region might be a good indicator of the review quality, not only for a well done review, but also for a badly done review. This might also help to assess the mental state of the reviewer on a region. For example, we could expect a reviewer that had spent too much time in a complex region to be mentally exhausted, which might influence the remaining review. Or the other way around, where a reviewer that spent less time on a complex review might indicate he did not give the necessary atention to the region. Nonetheless, this would require external information as there are some factors (i.e. expertise on the language) that are relevant to correctly assess the review.

Revisits might be a good indicator of expertise. Both expert volunteers had similar overall performance with one of them being able to indentify 1 more bug as shown in Table 4.1.

TABLE 4.1: Performance with revisits of expert volunteers

| | TP | FN | FP | Revisits |
|---|---|---|---|---|
| Expert Volunteer | 3 | 10 | 3 | 291 |
| Expert Volunteer Working on the field | 2 | 11 | 3 | 135 |

From analyzing Table 4.1 we can see the expert volunteer was able to find 1 more bug then the expert volunteer who works on the field. However, the latter had less then the average number of revisits in each code. For each region the expert volunteer working on the field would have 30% to 60% less reviews then the average.

When comparing both expert volunteers revisits, we can see that the expert volunteer working on the field had less 63% revisits. This might indicate that the expert volunteer with less revisits, has more experience.

Concerning the biometric features, and relying on our domain knowledge from the following papers [[30],[19]], it was shown that ratio between the low frequency and the high frequency of the heart rate is a good surrogate of the cognitive load. This feature was a good indicator of the cognitive load during a task. This feature was accurate on assessing the cognitive load of the regions from each code as it also matched the code complexity classification of the regions. Across all codes, the region with highest complexity was region 2 of the quickSortIterative code. This region had the most time spent on and also the highest LF/HF mean. Other HR features like RMSSD, SDNN also support this as they show the smallest mean value for the same region.

Following in Table 4.2 we can see the real performance of the volunteers. In the table the "True Positives" are the real bugs that were identified has bugs (the wanted outcome), the "False Negatives" are existing bugs that were not identified by the volunteers and the "False Positives" are pieces of code identified by the volunteers as bugs that were not actually bugs.

TABLE 4.2: Performance of the volunteers

| | Expertise | True Positives | False Negatives | False Positives |
|---|---|---|---|---|
| Volunteer 1 | Non-expert | 3 | 10 | 14 |
| Volunteer 2 | Non-expert | 3 | 10 | 5 |
| Volunteer 3 | Expert | 3 | 10 | 3 |
| Volunteer 4 | Expert | 2 | 11 | 3 |
| Volunteer 5 | Non-expert | 2 | 11 | 6 |

From Table 4.2 we can see that the expert volunteers had less false positives when compared to the non-experts. We can also see that volunteer 2, which was the volunteer who spent most time reviewing, was able to get a close performance to the expert volunteers.

Following the complexity of each code, Code 2 (iterative) would be the hardest code, followed by code 3 and code 1. It would be expected to have features like RMSSD (lower when the cognitive load is higher), to have smaller values in Code 2. We will firstly start by analyzing each code and comparing the 2 regions of that code.

In the following figure we can see both region 1 and region 2 from code 1. We split the data in regions in order to also compare the regions between themselves.

Figure 4.2 show the LF/HF Values. In the graphs we can see that different volunteers have different sample values. Sample values are higher the more time was spent in the region. When comparing code 1 and code 2, we can see that the sample values range is wider (more than two times bigger) meaning that the time spent in region 2 was bigger than region 1. Another aspect we can see when analyzing both graphs is that the LF/HF values scale is higher in region 2. Not only that, while in region 1 the max LF/HF in region 1 is around 350, in region 2 the LF/HF max is around 1400. In region 2 we can also see that several times volunteers present LF/HF peaks near 500. This suggests that region 2 was more mentally demanding then region 1 in code 1.

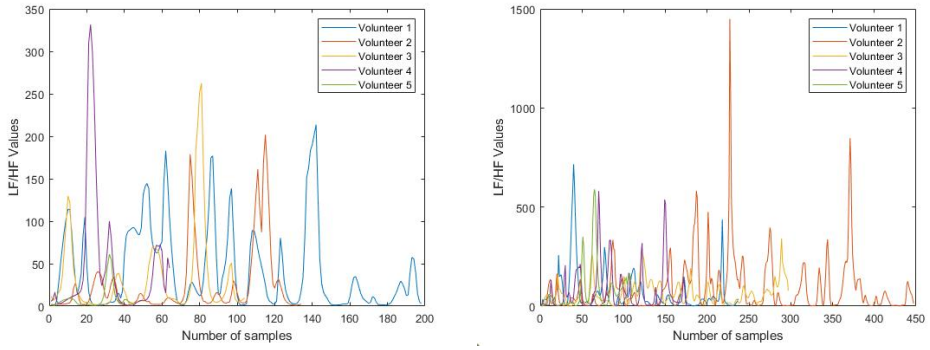


FIGURE 4.2: Values of LF/HF in region 1 (graph on the left) and region 2 (graph on the right) in code 1

In region 2 we can see that volunteer 2 (non-expert) had the highest peak and more number of samples which also means he spent the most time on this region. This is alarming, we would consider the volunteer to probably not have a good review. However, the volunteer was able to find one bug which matches the experts score. So we think that this volunteer needed more time to comprehend the code but still managed to review it.

In the following figure we can see both region 1 and region 2 from code 2.

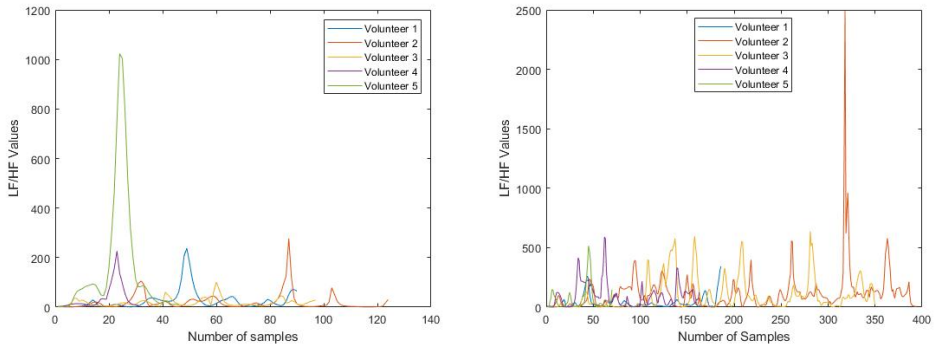


FIGURE 4.3: Values of LF/HF in region 1 (graph on the left) and region 2 (graph on the right) in code 2

Figure 4.3 show that in region 1 we can see a clear peak which might indicate that in this region there was some code the volunteer was struggling with. The LF/HF values were mostly under 200 which indicate a lower mental challenge when compared to region 2. We can also see that the sample width is quite small when compared to region 2 (nearly a quarter of region 2). In region 2 we have another huge peak reaching an LF/HF value of nearly 2500. This peak belongs to volunteer 2 again which is also the volunteer who takes the most time to do the review. Once again, this volunteer was able to find one bug which matches the experts score. This strengthens the idea that the volunteer needs more time to comprehend the code but still manages to review it.

In the following figure we can see both region 1 and region 2 from code 3.

Figure 4.4 shows that code 3 seems to be different than the remaining codes, as both of the regions seem to have closer scales when compared to the two previous

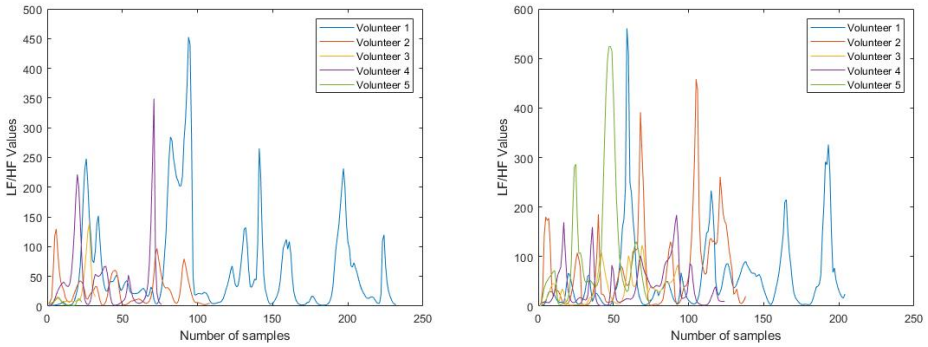


FIGURE 4.4: Values of LF/HF in region 1 (graph on the left) and region 2 (graph on the right) in code 3

codes. This suggests that region 1 of code 3 was harder then the previous regions 1 from code 1 and code 2. In fact, region 1 of code 3 is the only region 1 with at least 1 bug. Both regions also have similar number of samples, which indicate that the volunteers took similar amounts of time in both regions.

From a broader scope we could conclude that the data reflected the difficulty of the codes, as higher values of LF/HF were shown on the regions deemed more difficult.

As it was discussed previously on the Feature Engineering section, some features (e.g., RMSSD, SDSD, SDNN, S1S2) have similar behaviours when the reviewer is under a mentally demanding task. They tend to get smaller as the intervals between N-N beats get shorter. In Table 4.3 we show the mean values for relevant HR and EDA features to compare with the previous analysis using the LF/HF ratio.

TABLE 4.3: Table of means in the code snippets

| | HR | LF/HF | RMSSD | SDSD | SDNN | S1S2 | EDA | SCL |
|---|---|---|---|---|---|---|---|---|
| Code 1 | 73,7302 | 53,07235767 | 0,000444 | 0,000377 | 0,011399 | 0,004654 | 7,39454 | 5,177361 |
| Code 2 | 74,49466 | 67,48448656 | 0,000299 | 0,000255 | 0,0076 | 0,003103 | 11,5046 | 8,927945 |
| Code 3 | 73,90403 | 48,29276057 | 0,00041 | 0,000356 | 0,01013 | 0,004136 | 11,5176 | 8,801614 |

From analyzing Table 4.3, we can see that the HR and LF/HF have their highest values in code 2, which shows code 2 to be the most mentally demanding task. The remaining HR features (e.g., RMSSD,SDSD,SDNN and S1S2) also confirm the previous has they have the lowest values in code 2.

Analyzing EDA features we can see that the EDA mean and the skin-conductance level (SCL) are higher in code 2 and code 3, showing a different pattern then the HR features did.

By analyzing HR and LF/HF in code 1 and code 3 we can see that the values are quite close. Analyzing the remaining HR features confirms this assumption, as every HR feature has closer values when compared to code 3. This might indicate that code 1 and code 3 are similar in complexity, having being reviewed with similar mental effort. The calculated vg of code 1 is 6 and code 3 is 8, which also puts both code 1 and code 3 close in complexity.

Looking to the EDA feature, it might be a good indicator of stress. In our group of volunteers, one of the expert works in a related field. His EDA signals were lower from the remaining volunteers. The other expert volunteer also has a different EDA behaviour then the non-expert volunteers.

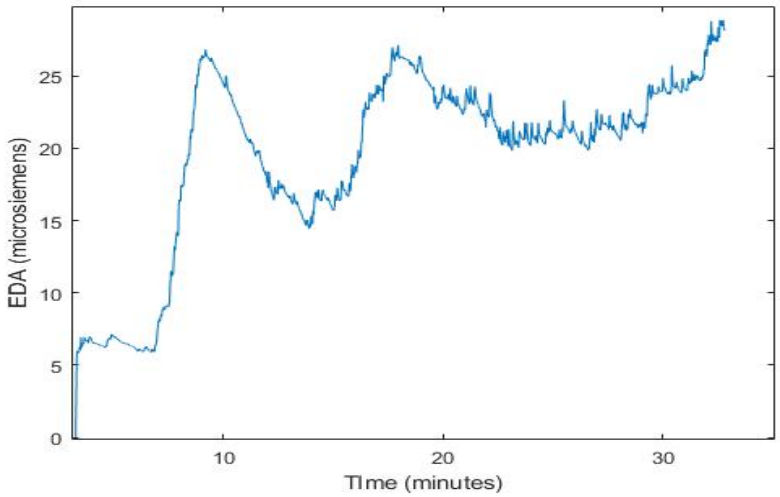


FIGURE 4.5: EDA over time of non-expert volunteer

Figure 4.5 shows an example of the EDA values (Y axis) of a non-expert volunteer. This range is also similar in the remaining volunteers. While reviewing the first code, which corresponds roughly to the first 10 minutes in Figure 4.5, this volunteer identified 10 bugs that were not bugs. In the two remaining codes the volunteer did also identified bugs there were non-bugs.

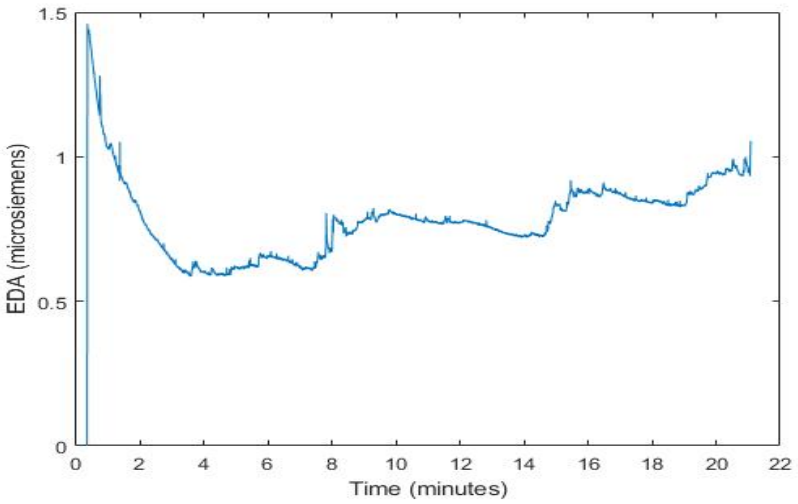


FIGURE 4.6: EDA over time of the expert volunteer working on the field

Figure 4.6 shows the EDA values over time of the expert volunteer that works in related field. Note that the EDA values (Y axis) goes up to 1.5, throughout the experiment.

The comparison between both EDA values might indicate a difference in stress while performing the review. As the expert reviewer that is already familiar with the process, he will be more comfortable resulting in lesser stress while performing the task, even though he finds bugs or not.

After applying an ML pipeline that consisted of feature selection, cross-validation, prediction, and performance metrics calculation, we were able to obtain good results by far. However, there were some limitations due to the relatively low number of samples. We could not use the Leave-One-Out cross-validation again due to precision dropping. Thus, for this reason, we used the K-Fold with K=5 cross-validation. In Table 4.4 are the results with two feature selection methods, the ANOVA variance, and the Chi-squared.

TABLE 4.4: Table of classifier results using ANOVA Feature selection and Chi-squared

| Feature Selection Method | Classifier | Accuracy | Precision | Recall |
|---|---|---|---|---|
| ANOVA Feature Selection | Logistic Regression | 68.7% ± 0.26 | 0.56 | 0.70 |
| | Linear Discriminant Analysis | 81.33% ± 0.219 | 0.73 | 0.70 |
| | K-nearest neighbor k=5 | 78.6% ± 0.14 | 0.76 | 0.83 |
| | Classification and Regression Trees (CART) | 62.0% ± 0.12 | 0.56 | 0.51 |
| | Naive Bayes | 82.0% ± 0.13 | 0.83 | 0.83 |
| | Support Vector Machine - Linear | 72.0% ± 0.09 | 0.60 | 0.83 |
| Chi-2 | Logistic Regression | 81.10% ± 0.18 | 0.73 | 0.70 |
| | Linear Discriminant Analysis | 72.00% ± 0.16 | 0.60 | 0.69 |
| | K-nearest neighbor k=5 | 72.00% ± 0.17 | 0.63 | 0.80 |
| | Classification and Regression Trees (CART) | 71.00% ± 1.23 | 0.80 | 0.50 |
| | Naive Bayes | 81.00% ± 0.22 | 0.76 | 0.90 |
| | Support Vector Machine - Linear | 62.00% ± 0.11 | 0.9 | 0.91 |

In Table 4.4 the best result is obtained in both Feature Selection Methods used (ANOVA Feature selection and Chi-squared) with the Naive Bayes classifier. With the ANOVA Feature Selection using the Naive Bayes Classifier the accuracy achieved was 82% with a precision of 0.83 and a recall of 0.83.

With the Chi-squared using the Naive Bayes Classifier the accuracy achieved was 81% with a precision of 0.76 and a recall of 0.90.

# Chapter 5

# Discussion and Future Directions

This thesis introduced the iReview tool concept, design, and preliminary architecture. The iReview tool is expected to help achieve better code review quality by evaluating its quality from the individual's standpoint. iReview evaluates the code review quality by assessing the code reviewer's cognitive load, representing the engagement level and code comprehension level while doing the review using non-intrusive biofeedback devices (e.g., smartwatches and bracelets).

The tool also uses a low-cost eye-tracker compatible with software development and code review settings. The eye-tracker provides pointers to the code regions that were not well-reviewed. By this evaluation and localization of the code regions, the tool advises the code reviewer or the code author to do a second review pass for the code regions that were not well covered.

This tool uses Artificial Intelligence to fuse biometric features (i.e., HRV and EDA) and non-biometric features, namely code complexity, review time, the number of revisits to the code region, and experience in one feature space. The features are fed into classifiers to classify each code region as either well or badly reviewed.

iReview makes use of widely available and commercial wearables to allow software companies to use the tool without burdening extra loads on the reviewers while doing the task.

In this thesis we assessed cognitive load during code comprehension tasks using HR and EDA. We also considered standard features (i.e. code complexity metrics, expertise) and used the NASA-TLX questionnaire which is used to perceive the workload in order to assess a task.

The results obtained suggest that it is possible to assess cognitive load using lightweight non-intrusive and low-cost sensors. We think that the tool presented in this thesis was a big first step in the field that might give the opportunity for researchers to continue the work. Not only that, it also opens doors for tools of the same kind but analyzing different content. For example, a similar tool could be used to assess the cognitive load while reading a medical report. The applications that a tool of this kind could are vast.

As future work for this tool, I consider that would be valuable to:

- Increase the number of volunteers
  This would allow for a deeper data analysis including the SCR feature. Specifically, the increase of expert volunteers with various levels of expertise would be interesting to analyze as we could see how different levels of expertise react during code review;

- Prepare and experiment in a real life scenario
  Experiment the tool in a real scenario would give rich data to analyze and also

feedback of how a day-to-day code reviewer would feel while working under the spotlight of the tool. This is very important has it is feedback from one of the main users of the tool;

- Implementation of the tool in version control tools
  As code reviews today in companies usually are done through the use of version control tools (e.g., Git, Mercurial), usually using platforms that manage these kind of tools (e.g., Github, GitLab, Bitbucket) it would be interesting to import the tool in order to make it available in platforms that are already being used by companies like GitHub, GitLab or so many others.

# Appendix A

# Appendix A

## A.1 Code Snippets

### A.1.1 Distance between primitive numbers

```
// TASK ST3.E.B - Largest distance between consecutive primes
#include <stdio.h>

void primeDist() {
    int from, to, n, i;   // loop control variables
    int prime = 1, numprimes = 0, sum = 0;
    int prev, dist;  // last prime and largest distance

    printf("\nEnter from - to values\n");
    scanf("%d", &from);
    scanf("%d", &to);
    prev = from;
    dist = 0;
    for (n = from; n <= to; n++) {
        // checks for primality
        for (i = 1; i <= n/2; i++) // could be optimized, but
that's ok
            if ( n / i == 0) {
                prime = 0;  // not prime after all
                break;
            }
        // in case of prime, checks distance from previous
        if (prime==1) {
            printf("%d  ", n);
            if (n - prev > dist)  // is this distance is larger,
then updates
                dist = n - prev;
            numprimes++;
            sum += n;
        }
    }

    printf("\nNumber of primes in %d - %d : %d", from, to,
numprimes);
    printf("\nLargest distance between primes is %d\n", dist);
    printf("\nSum of all primes is %d\n", sum);
}
```

FIGURE A.1: Code 1 of the protocol

### A.1.2 Quick Sort implementation

```
// TASK ST3.H.B - Quicksort iterative implementation
#include <stdio.h>

void quickSortIterative(int array[], int left, int right) {
    int stack[right - left + 1];  //  auxiliary stack for segment limits
    int top = -1;  // signals top of stack (empty stack)
    // push limits of initial segment into the stack
    stack[++top] = left;
    stack[++top] = right;

    while (top > 0) {  // while stack not empty, get next segment and process
it
        // Pop array segment limits
        left = stack[top--];
        right = stack[top--];
        // partition array segment [left,right] and finds pivot element
        int pivotval = array[right];
        int i = (left - 1);
        int tmp;
        // finds pivot position
        for (int j = left; j <= right - 1; j++)
            if (array[j] > pivotval) {
                i++;
                tmp = array[i];      // swaps array[i] with array[j]
                array[i] = array[j];
                array[j] = tmp;
            }
        // swap final array[i + 1] with array[right]
        tmp = array[i + 1];
        array[i + 1] = array[right];
        array[right] = tmp;
        int p = (i + 1);  // p is the pivot position

        // Push left-side segment to stack
        stack[++top] = left;
        stack[++top] = p - 1;

        // Push right-side segment to stack
        stack[++top] = p + 1;
        stack[++top] = right;
    }
}
```

FIGURE A.2: Code 2 of the protocol

### A.1.3 Remove Duplicated values from an array

```
// TASK ST3.M.B - Remove duplicate values from array
#include <stdio.h>
#define MAX_SIZE 100 // Maximum size of the array

void removeDups() {
    int arr[MAX_SIZE]; // Declares an array of size 100
    int size;          // Total number of elements in array
    int i, j, k;       // Loop control variables

    // Input size of the array
    printf("Enter size of the array : ");
    scanf("%d", size);
    if (size<1 || size >MAX_SIZE)
        size = MAX_SIZE;
    // Input elements in the array
    printf("Enter elements in array : ");
    for(i=0; i<size; i++)
        scanf("%d", &arr[i]);

    for(i=0; i<size; i++) {
        // searches for other occurrences of that value
        for(j=0; j<size; j++)
            // If any duplicate found
            if(arr[i] == arr[j]) {
                // delete the current duplicate element
                for(k=i; k < size - 1; k++)
                    arr[k] = arr[k + 1];
                // decrement size after removing duplicate element
                size--;
            }
    }

    // Prints array afterwards
    printf("\nArray elements after deleting duplicates : ");
    for(i=0; i<size; i++)
        printf("%d\t", arr[i]);
}
```

FIGURE A.3: Code 3 of the Protocol

| Goal | Activity | Due date | Deliverables | Milestones |
|---|---|---|---|---|
| Review the literature systematically for current state-of-the-art code reviews tools and techniques and data-driven approaches for assessing code comprehension/review. | a. Select 15-20 relevant recent papers, possibly with the following keywords (code review tools, code review cognitive load, code review quality, data-driven code comprehension assessment…etc.).<br>b. Summarize the overall idea, the methodology, the results, and the drawbacks.<br>c. Identify biometric features in other studies and compare them with your research.<br>d. Establish your benchmarking criteria to compare with other possibly similar studies/tools.<br>e. Report the literature review with comprehensive analysis and reflections. | 15-11-2021 | Introduction Chapter<br><br>Literature Review Chapter including the previously established biometric features and their correlation with the cognitive load and attention, also to the review quality | M1 |
| Design 3-4 code reviews protocols | a. Describe the tool functionalities, use-cases, and moods of operation.<br>b. Select realistic 10-15 code snippets with various complexities.<br>c. Recruit 10-15 subjects/volunteers knowing programming and code reviewing.<br>d. Schedule the experiments with the volunteers.<br>e. Approve the protocol with the team.<br>f. Perform the data acquisition using the sensors and the wearables (e.g., Fitbit watches).<br>g. Establish biometric baselines for each reviewer.<br>h. Include subjective questionnaires (e.g., NASA-TLX or others).<br>i. Report and visualize the readings | 1-10-2021<br>To<br>1-02-2022 | 3-4 protocols with realistic cases (you are recommended to consult SW companies)<br><br>A brief report showing the results (part of Chapter 3).<br><br>1-2 presentations at least. | M2 |
| Analyze Data | a. Pre-process data (e.g., cleaning, missing data imputation, normalization).<br>b. Select best features that have good correlations with the CL (based on literature and/or data-driven feature selection).<br>c. Report the features, their importance, correlation with CL, and code review quality.<br>d. Select the features time window to capture the CL (based on established literature and experimental data).<br>e. Prepare the training data and label them (we may use different ways of labeling).<br>f. Apply Quality Rules (refer to the TSE paper).<br>g. Train 3-4 classifiers (e.g., Decision Tree, KNN, Random Forest). | 1-2-2022<br>To<br>1-4-2022 | A clean dataset<br><br>3 presentations on the progress (monthly)<br><br>1 progress report<br><br>Last Part of Chapter 3<br><br>First part of Chapter 4 (Analysis) | M3, M5 |
| Design and deploy the prototype of the tool | a. Investigate various code portioning tools based on complexity, non-overlapping constructs…etc.<br>b. Adapt an automated code complexity calculator.<br>c. Design the review interface, which introduces the main functionalities (e.g., start recording data and baseline, start the review, pause review, highlight bugs, add comments, pause review, discard review, save the review, generate the quality report.<br>d. Design the quality report interface, which includes:<br>a. Overall quality evaluation (Good/Bad).<br>b. Indication of code regions that are (Bad).<br>e. Design and implement the interfacing with GitHub, including:<br>a. Sync the eye-tracker and the wearable data in an interface.<br>b. Integrate the interface in (a) with the GitHub platform. | 25-3-2022<br>To<br>1 week prior to the Submission date | Automatic code portioning methodology<br><br>Tool Interface<br><br>Report Interface.<br><br>Integrated tool<br><br>Chapter 5 (Results and Discussion) | M4, M5 |

TABLE A.1: Planning of Thesis

## Gantt chart

*February 8 to June 14*

Activity Duration

| ACTIVITY | PLAN START | PLAN DURATION | WEEKS PER MONTH | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | February | | | March | | | | April | | | | May | | | | June | | | | |
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | |
| Thesis writting | 1 | 19 | | | | | | | | | | | | | | | | | | | | |
| Research/Literature Review | 1 | 9 | | | | | | | | | | | | | | | | | | | | |
| Eye-tracker communication | 1 | 3 | | | | | | | | | | | | | | | | | | | | |
| Fitbit sense communication | 4 | 6 | | | | | | | | | | | | | | | | | | | | |
| Eye-tracker and Fitbit Sync | 7 | 8 | | | | | | | | | | | | | | | | | | | | |
| Development of a plugin | 3 | 11 | | | | | | | | | | | | | | | | | | | | |
| Protocol/Scenarios design | 8 | 13 | | | | | | | | | | | | | | | | | | | | |
| Experiments | 9 | 15 | | | | | | | | | | | | | | | | | | | | |
| Tool Interface Design | 8 | 12 | | | | | | | | | | | | | | | | | | | | |

FIGURE A.4: Gantt Chart on future works

# Bibliography


[1] Anupama Aggarwal. *July 22, 1962: Mariner 1 Done In by a Typo*. https://www.wired.com/2009/07/dayintech-0722/. 2009. (accessed: 22:13 16/11/2021).

[2] Autonomic Nervous System Anatomy. *Anatomy, Autonomic Nervous System*. https://www.ncbi.nlm.nih.gov/books/NBK539845/. [Online; accessed 16-January-2022].

[3] R.C. Atkinson and R.M. Shiffrin. "Human Memory: A Proposed System and its Control Processes". In: ed. by Kenneth W. Spence and Janet Taylor Spence. Vol. 2. Psychology of Learning and Motivation. Academic Press, 1968, pp. 89–195. DOI: https://doi.org/10.1016/S0079-7421(08)60422-3. URL: https://www.sciencedirect.com/science/article/pii/S0079742108604223.

[4] Alberto Bacchelli and Christian Bird. "Expectations, outcomes, and challenges of modern code review". In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 712–721. DOI: 10.1109/ICSE.2013.6606617.

[5] Ricardo Couceiro et al. "Biofeedback Augmented Software Engineering: Monitoring of Programmers' Mental Effort". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2019, pp. 37–40. DOI: 10.1109/ICSE-NIER.2019.00018.

[6] Ricardo Couceiro et al. "Biofeedback Augmented Software Engineering: Monitoring of Programmers' Mental Effort". In: *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. 2019, pp. 37–40. DOI: 10.1109/ICSE-NIER.2019.00018.

[7] M. E. Fagan. "Design and code inspections to reduce errors in program development". In: *IBM Systems Journal* 38.2.3 (1999), pp. 258–287. DOI: 10.1147/sj.382.0258.

[8] Michael E. Fagan. "Advances in Software Inspections". In: *Pioneers and Their Contributions to Software Engineering: sd&m Conference on Software Pioneers, Bonn, June 28/29, 2001, Original Historic Contributions*. Ed. by Manfred Broy and Ernst Denert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 335–360. ISBN: 978-3-642-48354-7. DOI: 10.1007/978-3-642-48354-7_14. URL: https://doi.org/10.1007/978-3-642-48354-7_14.

[9] Benjamin Floyd, Tyler Santander, and Westley Weimer. "Decoding the Representation of Code in the Brain: An fMRI Study of Code Review and Expertise". In: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, pp. 175–186. DOI: 10.1109/ICSE.2017.24.

[10] Thomas Fritz and Sebastian C. Müller. "Leveraging Biometric Data to Boost Software Developer Productivity". In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 5. 2016, pp. 66–77. DOI: 10.1109/SANER.2016.107.

[11] Davide Fucci et al. "A Replication Study on Code Comprehension and Expertise Using Lightweight Biometric Sensors". In: *Proceedings of the 27th International Conference on Program Comprehension*. ICPC '19. Montreal, Quebec, Canada: IEEE Press, 2019, 311–322. DOI: 10.1109/ICPC.2019.00050. URL: https://doi.org/10.1109/ICPC.2019.00050.

[12] Eduardo Gil et al. "Photoplethysmography pulse rate variability as a surrogate measurement of heart rate variability during non-stationary conditions". In: *Physiological measurement* 31 (Sept. 2010), pp. 1271–90. DOI: 10.1088/0967-3334/31/9/015.

[13] *Github and code-review*. https://github.com/features/code-review.

[14] Google. *Google Engineering Practices Documentation*. https://google.github.io/eng-practices/. (accessed: 11:07 18/11/2021).

[15] Google. *The Battle of the Bugs*. https://www.openrefactory.com/intelligent-code-repair-icr/. (accessed: 18:14 18/11/2021).

[16] Haytham Hijazi et al. "iReview: an Intelligent Code Review Evaluation Tool using BiofeedbacN". In: ().

[17] *How Many Software Developers Are There in the World?* URL: https://www.bairesdev.com/blog/how-many-software-developers-in-the-world/. (visited 09/11/2021).

[18] Becton Loveless. *Cognitive load theory - the definitive guide*. 2022. URL: https://www.educationcorner.com/cognitive-load-theory/.

[19] M. Malik et al. "Heart rate variability. Standards of measurement, physiological interpretation, and clinical use". English. In: *European Heart Journal* 17.3 (1996), pp. 354–381. ISSN: 0195-668X.

[20] Sage McEnery. *How much computer code has been written?* https://medium.com/modern-stack/how-much-computer-code-has-been-written-c8c03100f459. (accessed: 18:31 18/11/2021).

[21] *Meet Phabricator, The Witty Code Review Tool Built Inside Facebook*. https://techcrunch.com/2011/08/07/oh-what-noble-scribe-hath-penned-these-words/.

[22] Roberto Minelli, Andrea Mocci, and Michele Lanza. "I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time". In: *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension*. ICPC '15. USA: IEEE Computer Society, 2015, 25–35. ISBN: 9781467381598. DOI: 10.1109/ICPC.2015.12. URL: https://doi.org/10.1109/ICPC.2015.12.

[23] Sebastian C. Müller and Thomas Fritz. "Using (Bio)Metrics to Predict Code Quality Online". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016, pp. 452–463. DOI: 10.1145/2884781.2884803.

[24] Sebastian C. Müller and Thomas Fritz. "Using (Bio)Metrics to Predict Code Quality Online". In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016, pp. 452–463. DOI: 10.1145/2884781.2884803.

[25] N. Pinheiro et al. "Can PPG be used for HRV analysis?" In: vol. 2016. Aug. 2016, pp. 2945–2949. DOI: 10.1109/EMBC.2016.7591347.

[26] N. Pinheiro et al. "Can PPG be used for HRV analysis?" In: vol. 2016. Aug. 2016, pp. 2945–2949. DOI: 10.1109/EMBC.2016.7591347.

[27] *Review Board*. URL: https://www.reviewboard.org/integrations/. (visited 12/02/2021).

[28] Caitlin Sadowski et al. "Modern Code Review: A Case Study at Google". In: *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. ICSE-SEIP '18. Gothenburg, Sweden: Association for Computing Machinery, 2018, 181–190. ISBN: 9781450356596. DOI: 10.1145/3183519.3183525. URL: https://doi.org/10.1145/3183519.3183525.

[29] Janet Siegmund et al. "Understanding Understanding Source Code with Functional Magnetic Resonance Imaging". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: Association for Computing Machinery, 2014, 378–389. ISBN: 9781450327565. DOI: 10.1145/2568225.2568252. URL: https://doi.org/10.1145/2568225.2568252.

[30] Soroosh Solhjoo et al. "Heart Rate and Heart Rate Variability Correlate with Clinical Reasoning Performance and Self-Reported Measures of Cognitive Load". In: *Scientific Reports* 9 (Oct. 2019), pp. 1–9. DOI: 10.1038/s41598-019-50280-3.

[31] Davide Spadini et al. "When Testing Meets Code Review: Why and How Developers Review Tests". In: *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 2018, pp. 677–687. DOI: 10.1145/3180155.3180192.

[32] John Sweller. "Cognitive load during problem solving: Effects on learning". In: *Cognitive Science* 12.2 (1988), pp. 257–285. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(88)90023-7. URL: https://www.sciencedirect.com/science/article/pii/0364021388900237.

[33] *The battle of the bugs*. URL: https://www.openrefactory.com/intelligent-code-repair-icr/.

[34] Héctor Adrián Valdecantos et al. "An Empirical Study on Code Comprehension: Data Context Interaction Compared to Classical Object Oriented". In: *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 2017, pp. 275–285. DOI: 10.1109/ICPC.2017.23.