LUND UNIVERSITY

Toward Gaze-enabled Programming Tool Assistance

Kuang, Peng

2024

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Kuang, P. (2024). *Toward Gaze-enabled Programming Tool Assistance*. Lund University.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 08. Nov. 2026

Toward Gaze-enabled Programming Tool Assistance

Peng Kuang

Licentiate Thesis, 2024

Department of Computer Science
Lund University

LU-CS-DISS: 2024-2
ISBN 978-91-7623-305-4 (printed version)
ISBN 978-91-7623-306-1 (electronic version)
Licentiate Thesis 2, 2024
ISSN: **1652-4691**

Department of Computer Science
Lund University
Box 118
SE-221 00 Lund
Sweden

Email: peng.kuang@css.lth.se
WWW:
https://portal.research.lu.se/sv/persons/peng-kuang

Printed in Sweden by Tryckeriet i E-huset, Lund, 2024
Typeset using LaTeX
© *2024 Peng Kuang*

ABSTRACT

Programming is a cognitively demanding exercise. In particular, today's software development requires a collective effort of programmers and the orchestration of a complex programming infrastructure. As disruptive technologies emerge, e.g., AI and quantum computing, the programming practice is undergoing a change, facing an uncertain future that we may not be able to accurately predict but can envision and work toward.

With the maturity of eye-tracking and its integration into everyday consumer electronics such as Alienware's laptops and Apple's Vision Pro, we expect it will eventually make its way into everyday use just as touchpad, camera, and microphone. Therefore, we see an opportunity to design eye-tracking based assistance to support programmers. Given programmers spend a large amount of their time reading and understanding code, which heavily relies on eyes, we deem this to be a promising problem domain where eye-tracking can be of assistance.

To explore this inquiry, we undertook two mapping studies to establish the problem and solution constructs. We then surveyed professional developers to understand this representative cohort of our prospective users and gather concrete, situated problems from them. We conducted these studies under the guiding design science model for empirical software engineering which centers on a problem-solution pair.

From the first study, we found that eye-tracking so far is used mostly for education-oriented studies in the research community focused on software development. There is a need to bring it closer to practitioners. From the second study, we identify that the gaze data produced by eye trackers has been explored with a collection of machine learning techniques. However, these models were trained with small samples that might carry bias and insufficiency. Contemporary machine learning techniques may be able to compensate for that. From the survey, we learned that developers have already adopted AI assistance, and they are mostly positive about it despite room for greater accuracy and capability. As eye-tracking is relatively novel to them, most developers are unsure about how it can help them.

For future work, we plan to practice designing with programmers to develop and evaluate our proof of concept and explore gaze data with more tailored machine learning techniques, which aims to generate integration into our system.

ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervisors, Emma Söderberg, Martin Höst, and Diederick C. Niehorster for choosing to hire me and work with me. The support, patience, autonomy, and trust that they offered me throughout my research journey have been invaluable. I greatly appreciate their sharing of intelligence, experiences, and resources during our collaboration. From them, I have learned so much in order to become a qualified researcher and a good academic.

Special recognition goes to my main supervisor, Emma, who supports me not just in academics but also in other aspects. She has always been open to feedback, listening to my needs and wants, and making things happen, e.g., helping me acquire a diverse skill set to fulfill my career goals.

I want to thank Christoph Reichenbach for his compassionate friendship during challenging times. I am also thankful to Gareth Callanan, Qunying Song, and Daniel Helgesson, who befriended me from the beginning, easing my transition to a new country or a new stage of life. To Idriss Riouak, Alan McCabe, Anton Risberg Alaküla, Lo Heander, Andreas Bexell, and everyone (including the master students and alumni) in the SDE research group and NEX group, and many other colleagues in Administration and from other research groups of the department, I thank you all for your generous help, kind support, and good company. Furthermore, I want to thank the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231) for funding my employment at LU and Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation for involving me in a series of insightful courses and academic activities.

Finally, I dedicate this licentiate thesis to my late father Kuang Yugui, who passed away on 2 April 2022. Dad, your memory is cherished, and I forever hold you in my heart. I am deeply indebted to my mother Hu Nyusui, my elder sisters Kuang Tao and Kuang Li, and my elder brother Kuang Lin. To my beloved nephews Cao Xuan and Xie Rui, and niece Cao Qian, I'm so blessed to have each of you in my life.

Peng Kuang
Lund, 2 April 2024

LIST OF PUBLICATIONS

Publications included in the thesis

I **Toward Gaze-assisted Developer Tools**
Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst
Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER), 2023.

II **Applying Machine Learning to Gaze Data in Software Development: a Mapping Study**
Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst
Proceedings of the International Workshop on Eye Movement in Programming (EMIP), 2023.

III **Developers' Perspective on Today's and Tomorrow's Programming Tool Assistance: A Survey**
Peng Kuang, Emma Söderberg, and Martin Höst
Proceedings of the 10th Edition of the Programming Experience Workshop (PX), 2024.

Related Publications

VI **Visual Cues in Compiler Conversations**
Alan T. McCabe, Emma Söderberg, Luke Church, Peng Kuang
Proceedings of the 33rd Annual Workshop of the Psychology of Programming Interest Group Annual Workshop (PPIG), 2022.

VII **Ironies of Programming Automation: Exploring the Experience of Code Synthesis Via Large Language Models**
Alan T. McCabe, Moa Björkman, Joel Engström, Peng Kuang, Emma Söderberg, Luke Church
Proceedings of the 8th International Conference on the Art, Science, and Engineering of Programming, 2024.

CONTRIBUTION STATEMENT

The following papers are included in this dissertation:

Paper I Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. "Toward Gaze-assisted Developer Tools". In *Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. Melbourne, Australia, 2023. DOI: 10.1109/ICSE-NIER58687.2023.00015.

A replication package, including the data set and coding results of each round, was submitted for the paper at Lund University: portal.research.lu.se/files/137278187

Paper II Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. "Applying Machine Learning to Gaze Data in Software Development: a Mapping Study". In *Proceedings of the International Workshop on Eye Movement in Programming (EMIP)*. Tubingen, Germany, 2023. DOI: 10.114 5/3588015.3589190.

A replication package, including the data set and coding results of each round, was submitted for the paper at Lund University: portal.research.lu.se/en/publications/applying-machine-learning-to-gaze-data-in-software-development-a-

Paper III Peng Kuang, Emma Söderberg, and Martin Höst. "Developers' Perspective on Today's and Tomorrow's Programming Tool Assistance: A Survey". In *Proceedings of the 10th Edition of the Programming Experience Workshop (PX)*. Lund, Sweden, 2024.

A replication package, including the survey questions, the Python script used for quantitative analysis, and the code book used for qualitative analysis, was submitted for the paper at Zenodo DOI: 10.5281/zenodo.10777303.

The table below indicates the contributions Peng Kuang made to each paper:

Paper	*Design*	*Execution*	*Analysis*	*Writing*	*Replication Package*
I	◐	◐	◕	◐	◕
II	◕	◕	◐	◕	◕
III	◕	◐	◕	◕	●

The headers of the columns correspond to the different stages/tasks of our studies in sequential order.

The dark portion of the circle represents the amount of work and responsibilities assigned to Peng Kuang for each individual step:

- ○ No contribution
- ◔ Peng Kuang was a minor contributor to the work
- ◐ Peng Kuang was a contributor to the work
- ◕ Peng Kuang led and did a majority of the work
- ● Peng Kuang led and did almost all of the work

CONTENTS

Introduction **1**
1 Introduction 1
2 Background 4
3 Method 8
4 Contributions 11
5 Ethical Aspects 13
6 Threats to Validity 15
7 Conclusions and Future Work 16
References 18

Included papers **29**

I Toward Gaze-assisted Developer Tools **31**
1 Introduction 31
2 Method 32
3 Results 35
4 Discussion 37
5 Conclusions 39
References 41

II Applying Machine Learning to Gaze Data in Software Development: a Mapping Study **49**
1 Introduction 49
2 Method 50
3 Results 53
4 Discussion 58
5 Conclusions 60
References 61

III Developers' Perspective on Today's and Tomorrow's Programming Tool Assistance: A Survey **67**
1 Introduction 68
2 Method 69
3 Results 70
4 Discussion 83
5 Conclusions 84
References 86

INTRODUCTION

1 Introduction

In this section, we take a glance at the major milestones in the history of programming. We briefly review the changes in the profiles of human actors who perform programming and the notations and other media they leverage to communicate with computing machines. We highlight how modern software development practice evolved into its current shape and how the underlying tool set has expanded. Looking at the trajectory of programming, we observe that when revolutionary technology emerges, e.g., microprocessors in hardware and the incoming quantum computer, it may pose an impact on the entire programming world, potentially resulting in certain elements being deprecated, replaced, or reinvented. We discuss that we are now in an era of being disrupted by AI and many new assistive sensory technologies being experimented with(e.g., [8,91]). With the uncertainty that accompanies such disruption, we see an opportunity to design eye-tracking based tool assistance to support programmers, e.g., with code comprehension, given the fact that at the moment it consumes programmers most of their time [57, 103].

1.1 Changes in Programming Languages

In essence, programming is to instruct computers to perform tasks for humans. On the fundamental level, a computer only understands instructions consisting of binary digits. To ease the work of programming, computer scientists have developed various notations for expressing instructions. The notations have evolved from machine code and assembly language to now more human-readable high-level programming languages - C, Java, Python, and Rust, to name a few [18]. Although some are visual (e.g., Scratch [27]) or in between these two formats (e.g., Ballerina [20]), most programming languages are textual.

1.2 Changes in the Way of Programming and Programmer Population

As computers (e.g., microprocessors [21]) and programming languages are evolving, the way that programmers perform programming has also shifted. For instance, clerks used to heavily rely on punched cards for data processing [26] and operators used to control the state of the registers and memory of a computer and to debug a running program through front panels [19]. These practices are distinctly different from how programmers handle those tasks now. Nowadays, the common practice is to employ database management systems (e.g., MySQL) to store and process data. In most cases, we dedicate the operating system to interact with the registers and memory. For debugging, there are even more tools available to assist us such as debuggers (e.g., Google Chrome DevTools, the GNU debugger), unit test frameworks (e.g., JUnit), and static analyzers (e.g., SpotBugs).

Meanwhile, the programmer population has also changed. The programming team working with the first programmable, electronic, general-purpose, digital computer ENIAC consisted of six women [25]. Until the late 1960s, the general demographics of programmers were predominantly women in UK and US [9, 30, 42]. Nowadays, women's participation in computing varies in different countries [37, 60]. It is reported that girls make up roughly half of the students who enrolled in computing-related majors at some of the prestigious universities in India (Year 2014-2015 [88]), Malaysia (Year 2010-2018 [65]), and Saudi Arab (Year 2014 [6]). But less than half of the jobs in Information and Communication Technology are taken by women in many other countries, e.g., 22% in Sweden (Year 2019-2021 [86]), 20% in Australia (Year 2021 [63]), 22% in Nigeria (Year 2021 [36, 89]), and 35% in Argentina (Year 2009 [106]). Additionally, the main target segment of the visual programming language Scratch is children [27], a programmer profile that traditional programming language designers might have never had to consider.

In summary, significant changes have taken place over the past decades in terms of what concrete activities programming consists of and who practices programming.

1.3 Software Development and Tooling Today

Because of the advances in computing, software has carved its way into the workplace and later on became an indispensable component of our everyday life alongside the proliferation of personal computers [24] and the prosperity of the Internet [23].

With the rapid increase in computing power, however, a "software crisis" emerged in the 1960s [28]. Programmers at the time could not keep up with the increasing complexity of computer hardware and the software development tasks became larger and complex as well [78]. Many software development projects

were over-budget, over-time, inefficient, with low quality or low maintainability, or even never delivered.

These problems motivated the introduction of new software development processes, methodologies, and paradigms, which helped shape the mainstream programming practices today (e.g., Agile, DevOps). Accordingly, the underlying developer tooling has also evolved to now comprise editors (e.g., Sublime Text), assemblers/linkers/compilers (e.g., GCC for C++), build tools (e.g., Gradle), version control systems (e.g., Git), code review tools (e.g., Gerrit), testing tools and debuggers (e.g., Valgrind), and CI/CD tools (e.g., Jenkins), or become gigantic all-in-one Integrated Development Environments (IDEs, e.g., Visual Studio Code).

Undoubtedly, the size of the code bases or the number of lines of code that programmers work with also grew significantly if not exponentially. For instance, Google's monolithic code base contains billions of lines of code [69] and tens of thousands of developers use it as the single source of truth [73]. Several studies found that programmers spend most of their time on code comprehension, e.g., approximately 60% of their working hours [103], and 70% of their time in an IDE [57].

1.4 Future Programming and Programming Tool Assistance

Eye-tracking technology has matured significantly in the past few decades. Dedicated eye trackers are much more affordable and easier to use, and the more accessible web cameras can also achieve satisfactory performance. The laptop manufacturer Alienware has integrated eye-tracking into its high-performance models to enhance the gaming experience for the customers [91]. Apple has also released the visionOS platform which introduces a three-dimensional interface - eye movement, hand gestures, and voice commands - for some productivity products accompanied [8]. With these major players from consumer electronics entering the field to educate their clients on eye-tracking, we can be optimistic that eye-tracking will gradually become an integral sensor to many devices and systems or tools in the future, just as touchpad, camera, microphone, keyboard, and mouse. In this direction, some researchers have already been experimenting with enabling navigation and cursor with gaze in a software development environment, e.g., EyeDE [39], EyeNav [70], iReview [44], and Gander [75].

The release of ChatGPT has triggered a vast interest in Large Language Models (LLMs) or Generative AI among researchers and the public. It further drove the birth of the so-called AI software engineer or autonomous software engineering that is enabled by language models customized for software development tasks on the repository level, e.g., DevIn [4], SWE-agent [104], and AutoCodeRover [110]. Some of these intelligent agents are claimed to be teammates working with developers but not to replace them. Nonetheless, studies show developers have already started using AI assistance for programming, e.g., for repetitive code generation

and syntax recall [52]. There is also increasingly more research on "prompt engineering" with LLMs [54, 56, 100, 101] and some in particular aim to benefit beginner programmers or non-expert programmers [55, 108]. With the disruption of Generative AI or LLMs, e.g., ChatGPT [64], the possibility of using natural languages for programming has arisen [76,90]. We can speculate that the programmer profile may expand to a more diverse population since natural languages for programming are much more accessible, especially if one can program in one's native language. This implies that the underpinning infrastructure for programming may undergo a change and there is room for non-conventional assistive support to come into play as LLMs evolve toward multimodality [102].

Therefore, we see value in exploring how eye-tracking will fit into software development tooling given programmers spend a large amount of their time reading code to gather information from this abstract representation. This leads us to formulate the research question as follows:

- *RQ:* How can eye-tracking support programmers?

2 Background

In this section, we provide some fundamental concepts, practices, and related works as the background.

2.1 Eye Tracking

Eye tracking is the technology to capture the position and movement of one's eyes [58]. An eye tracker is a device that utilizes such technology to infer where or on what one's gaze is placed. As the most used modern model, a video-based eye tracker usually comprises three key components: near-infrared light illumination modules, a camera sensor, and a processor [58]. The illumination modules emit lights to help locate the target's eyes. The camera takes pictures of those eyes. The processor processes those images to model the eyes and map the positions of the eyes and the angles of the reflection into one's gaze points on a stimulus [33, 92], e.g., a screen. There are different ways to categorize eye trackers. Some manufacturers [58] assign them into three types: screen-based (since it is usually mounted to a desktop or laptop screen, as shown in Figure 1), wearable (e.g., glasses), and integrated (e.g., into a Virtual Reality headset). From the research perspective concerning experimental setup and human interaction, they can be characterized as head-free (e.g., wearable eye trackers), head-boxed (e.g., remote eye trackers), and head-restricted (e.g., eye trackers with a chin rest or forehead rest), according to the degree of the freedom of head movement [92].

Figure 1: A video-based eye tracker attached to the bottom of a monitor with a chin rest.

2.2 Gaze Data

As explained by Holmqvist et al. [45], we can measure the eye movement events recorded by an eye tracker in many ways and with great granularity. They suggested mapping the measures into four themes: movement (e.g., saccade direction), position (e.g., pupil diameter), numerosity (e.g., fixation number, proportion, and rate), and latency (e.g., entry time to an Area of Interest (AOI)) and distance (e.g., left eye to right eye).

Some researchers otherwise suggest categorizing gaze data into four orders [81]. The first-order gaze data comprises the X and Y positions, pupil diameter, and blink rate. It is usually the raw output of an eye tracker. Significant variation in pupil diameter usually indicates visual arousal such as an animation in a graphic interface but it can also be triggered by changes in lighting. Researchers are advised to adopt a high degree of rigor when collecting this measurement and interpret it cautiously. Blink rate can be a predictor of fatigue or drowsiness, e.g., the overload of a programmer's cognitive effort during extreme programming hours. Fixations and saccades are the second-order gaze data. They are probably the most commonly used gaze metrics. They constitute the foundation for most gaze analysis in

practice. Metrics directly derived from fixations and saccades are third-order gaze data, including fixation count, fixation duration, percentage of fixation, saccade count, saccade amplitude, and regression rate, just to name a few. They are useful for more fine-grained analysis and comparisons. For instance, both long fixation duration and large saccade counts on a specific code block of a file can indicate to which the programmer paid significant attention. The percentage of fixation can help distinguish the importance between two AOIs and the averaged fixation duration may help determine which one is more interesting to a programmer between two code snippets. The fourth-order gaze data refers to scan paths that consist of fixations or AOIs ordered by time. Scan paths are useful, for instance, for studying a programmer's distribution of visual effort over a large file in-depth or comparing two groups of programmers' search strategies such as whether a salient linear pattern is shown in reading the code.

2.3 Use of Eye Tracking in Software Engineering

There exists a wealth of Empirical Software Engineering (ESE) research using eye tracking [50, 62, 82]. The most studied topic to date is program or code comprehension (e.g., [13,31,32,67]). Other topics cover model comprehension (e.g., understanding UML diagrams [15, 107]), code review (e.g., [10, 11, 46]), debugging (e.g., [7, 98]), traceability (e.g., [79, 97]), data structure manipulation (e.g., [80]), and so on. Most of these studies focus on understanding programmers' eye movement behavior or speculating about their cognitive process by analyzing their gaze while performing certain software development tasks mentioned above. A great portion of these studies were undertaken with pedagogical purposes by comparing different groups of students (e.g., low vs. high performance, freshmen vs. graduates) or students with professionals (novices vs. experts). A few studies explored the possibility of introducing gaze-based support to programmers in their development environments. Within these studies, a substantial part of them employed gaze for human-computer-interaction (e.g., [39, 70, 74]) and sometimes together with voice (e.g., [66, 87]); some studies tracked programmers' gaze to inform them about where their attention was (e.g., [2, 75]) and further to facilitate them with completing a task with better quality (e.g., [44]) or efficiency (e.g., [16]).

2.4 Gaze Data, Machine Learning and Programming

As explained by Holmqvist et al. [45], the representation of gaze data is complex. For instance, attention maps are a well-adopted representation of the spatial distribution of gaze data. The representation can be visualized as a quite intuitive heat map. Both researchers and naive users can grasp a meaning from it easily. A weakness of a heat map is the loss of temporal information about the user. This illustrates the situation that researchers often face when conducting gaze data analysis. They have to select the representations that suit their study best from among

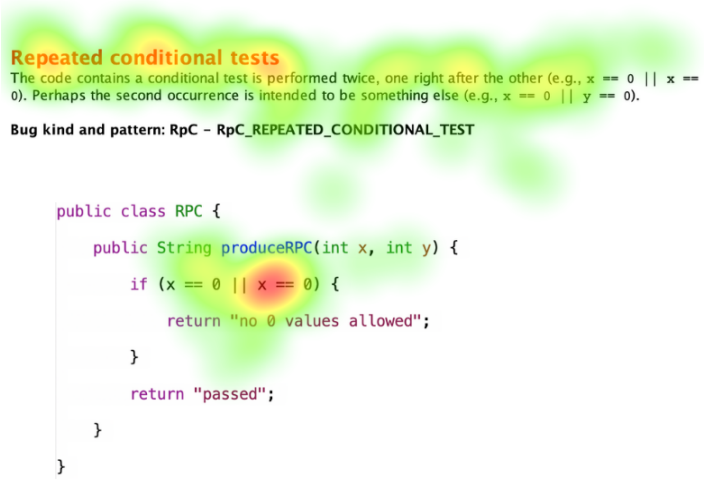


Figure 2: A heat map generated from a participant's reading of a program analysis example.

many options and deal with the associated trade-offs. Not every researcher is capable of or confident in making the right decision although there are some best practices to follow. And a bad choice will potentially render the validity of their findings in question.

On the other hand, machine learning (ML) is good at handling large amounts of data with high dimensionality to extract useful information from it, e.g., infer the category of the samples. Therefore, feeding the ML models with the raw gaze data which usually contains thousands of rows and tens of dimensions seems to be a good fit. It can potentially reveal details about the cognitive process and emotional state of programmers more precisely than the conventional representation and analysis. Further, when an ML model is well-trained with gaze data, it has the potential to be integrated into a programming environment to support programmers in real time while conventional representations are usually post-hoc.

A few studies have been conducted in the above-stated direction in the software engineering research community using eye tracking. Most of these studies focused on tackling the problem of code comprehension (e.g., [1,3,41,51]), along with code review [43, 96], pair programming [94], and debugging [109]. To approach the solution, they employed different machine learning techniques (e.g., support vector machines [3, 38, 51], random forests [5, 96]) to infer programmers' attributes (e.g., programming expertise [3, 5, 51], code reading ability [41]), the code's attributes (task difficulty [38], mentally demanding code fragments [1]), and the work's attributes (e.g., quality of code review [43], success of pair pro-

gramming [94]). Most of these works were done in the preliminary phase. It means that their functions were not integrated into a programming environment and empirically evaluated. To the best of our knowledge, however, some researchers (e.g., [44]) are pioneering in this direction.

3 Method

In this section, we first illustrate the overarching methodology that guides our studies. We then break down our high-level RQ explicated in the introduction into three sets of sub-RQs, which we have investigated through three separate studies, respectively. We briefly introduce the specific research methods that we employed in each of these three studies. Finally, we map these studies into the guiding Design Science model.

3.1 The Design Science Model

The methodology we followed is the Design Science model for Empirical Software Engineering proposed by Runeson et al [34, 71]. In their view, Empirical Software Engineering research mostly fits into the paradigm of "Design Science" [85]. This paradigm centers the frame of a "problem-solution" pair [93]. Most ESE studies encompass or at least constitute part(s) of the loop - problem conceptualization, solution design, and empirical validation. ESE researchers output knowledge through "problem-solution" instances and "technological rules" to benefit practitioners and peers. We detail their proposed model by briefly explaining each of the components. As shown in Figure 3, the model is divided into four quadrants. The horizontal axis divides the top and bottom into theory and practice. The vertical axis divides the left and right into problem and solution domains.

We first look at the four major box clusters in the model from top left to bottom left in a clockwise order.

- *Problem Construct* is the knowledge about a general problem, e.g., reading code is cognitively demanding and slow.
- *Design Construct* is the knowledge about a general solution, e.g., consistent style of the code facilitates code reading.
- *Problem Instance* is a specific problem, e.g., locating the critical code elements such as the variables and methods within a file is difficult and time-consuming.
- *Design Instance* is a concrete solution, e.g., syntactically highlighting these code elements with consistent colors makes it easier and quicker to locate them.

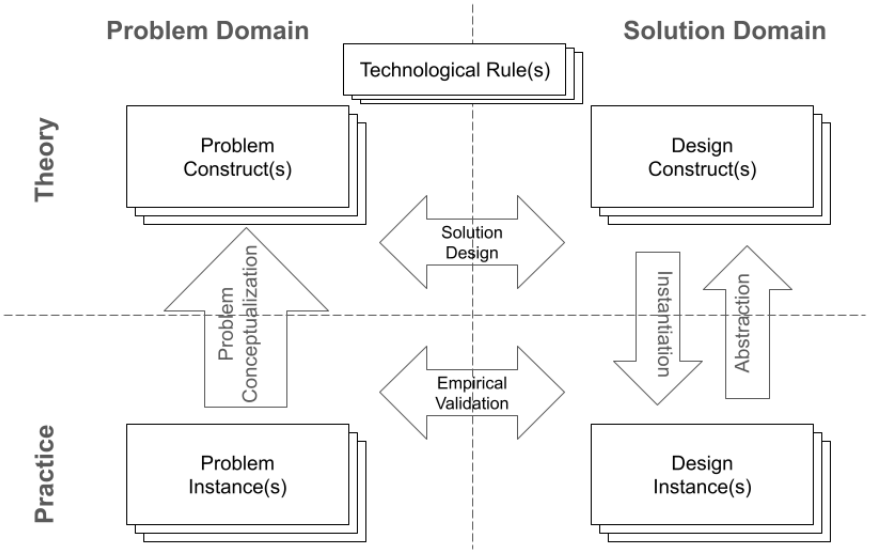


Figure 3: The Design Science model for ESE research, adapted from Runeson et al [71].

The plural expression and cluster format imply the outputs can be iterative and thus many.

Next, we interpret the knowledge-creating activities (the arrows) sitting between the four box clusters. We start the interpretation from problem conceptualization in a clockwise order as it is typically the first step of a design science study.

- *Problem Conceptualization* is to identify a problem, e.g., a field study with practitioners can identify a problem.
- *Solution Design* is to match a problem with a solution, e.g., a co-design workshop can map a solution to a problem.
- *Instantiation* is to actualize a solution in context, e.g., developing a prototype to address a specific problem.
- *Abstraction* is to extract key design decisions that served the purpose of the actualized solution, e.g., generalizing learning from testing a prototype with prospective users.
- *Empirical Validation* is to assess to what extent the actualized solution tackles the problem, e.g., conducting user studies with the target group.

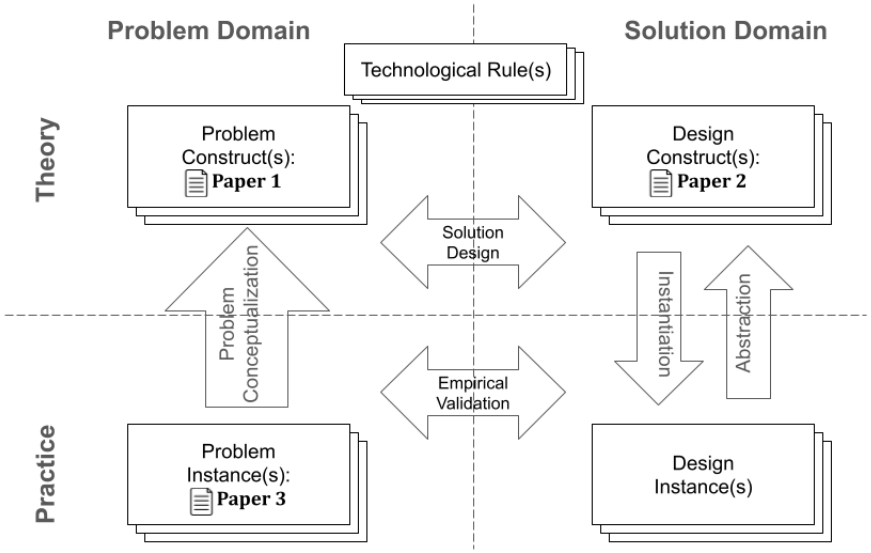


Figure 4: Our papers mapped to the Design Science model for ESE research, adapted from Runeson et al [71].

Lastly, we explain the small box cluster of technological rules.

- *Technological Rule* is the abstract knowledge built through conducting the study, e.g., utilizing syntax highlighting can ease the reading of code in a file and help locate the critical code elements more quickly.

3.2 Papers Mapped to the Design Science Model

Bearing the design science paradigm in mind, we have started our heuristic search [85] to answer the RQ: How can eye-tracking support programmers? We break down this broad inquiry into three pieces in the problem and solution space.

In Paper I [48], we conducted a mapping study to navigate the landscape of software development and eye tracking. A mapping study (a.k.a. scoping study) is a research method used to gain an overview of an area of interest by classifying and counting the related publications [68]. With this study, we wanted to understand:

- *RQ1.1:* How is eye tracking used in software engineering research?

In Paper II [48], we conducted another mapping study on software development, eye tracking, and machine learning to narrow down the problem and shape the solution. We intended to examine:

- *RQ2.1:* What problems have been tackled by using ML on gaze data in software development?
- *RQ2.2:* How is ML used with gaze data in software development?
- *RQ2.3:* What are the open challenges in using ML on gaze data in software development?

In Paper III [49], we surveyed professional programmers to build an in-depth understanding of programming practices in the industry. According to Linåker et al., survey is a method to "collect information about a group of users by sampling individuals from a large population" [53]. We sampled this vital but less-studied (in our field and due to the fact it is expensive to gain access to the group) cohort of our prospective users to gather their experiences, gauge their attitudes, and collect concrete pain points as in-context problems. We investigated the following research questions:

- *RQ3.1:* What are developers' perception on tool assistance today?
- *RQ3.2:* What are developers' perception on directions for future tool assistance?

We map our studies into the clusters of this model according to their main fits as shown in Figure 4. Paper I constitutes Problem Constructs. It first establishes the context that eye tracking has been used mostly for understanding students' gaze behavior during code comprehension tasks. It surfaces the problem that novice programmers struggle to read and understand code effectively. Paper II constitutes Design Constructs. It informs us that machine learning together with gaze data has the potential to predict a range of interesting attributes about the programmer, the code, or the software development task that one is undertaking. Such information can be used to inform the system to tailor its tool assistance for the designated programmer. Paper III constitutes Problem Instances. It reports that code comprehension is the most outstanding pain point even among experienced practitioners in the industry, wrapped with concrete information about their specific situations.

4 Contributions

In this section, we outline the contributions of each of our three studies to the research community.

4.1 Contribution of Paper I

In Paper I [50], we conducted a mapping study to examine the literature on using eye-tracking for software development studies. Answering RQ1.1, we identify that

the majority of the studies were conducted with students, with educational materials, and in the lab setting, and adopted the quantitative research method. We also recognize challenges in characterizing programmers' expertise due to its complex nature, and in building cross-study knowledge due to the inherent vulnerability of eye-tracking and various ways of performing gaze analysis.

To tackle these gaps and challenges, we make actionable recommendations to the wider community. We point out the need for more research with higher realism, namely, to study practitioners, with realistic code, and in real-world workplaces. We encourage researchers to involve practitioners in design studies if the design intends to monitor or detect the programmers' cognitive process or emotional state since this qualitative approach curates more understanding about the prospective users. As expertise is complex and fluid and gaze data is of huge amount, we suggest exploring gaze data for expertise inference with contemporary machine learning techniques.

4.2 Contribution of Paper II

In Paper II [48], we conducted a mapping study to particularly investigate how gaze data gathered from software development tasks are explored with specific machine learning techniques. The following findings with respect to problems tackled and ML technique utilized address RQ2.1 and RQ2.2. We found only a small portion of eye-tracking studies in empirical software engineering focus on such a topic. These studies, however, have covered most of the important software development tasks: code comprehension, code review, pair programming, and debugging. A wide range of ML techniques have also been utilized while support vector machines and random forests are the most frequently used ones. Researchers have applied ML to gaze data in software development to predict programmers' expertise, the difficulty of the task at their hands, and the quality of the work done by them (e.g., quality of code review).

To answer RQ2.3, we present the open challenges as follows. Despite the fact that the research community collaborated to curate open datasets, we found most studies trained their ML models with less than 40 samples (here we mean the number of distinct participants not the number of data points). This limitation, together with the lack of diversity and balance in the training dataset, calls into question the applicability or generalizability of these models. Further, some other problems, e.g., oversimplification of the programming expertise (which manifested in making the prediction task become a binary categorization), would have also impacted the accuracy. We also summarize the possible future works from these studies.

4.3 Contribution of Paper III

In Paper III [49], we surveyed developers to gather their experiences with and opinions on the current programming tool assistance and their attitudes toward

three enabling technologies (AI/ML, eye tracking, and gamification) for future programming tool assistance. We received responses from 68 developers who reside in 12 countries and speak 13 native languages.

To answer RQ3.1, we summarize the developers' perspectives on current programming tool assistance as follows. We found only half of these developers use program analysis but those who do think it is useful. Many of these developers have already used AI-based tools in their day-to-day work and a majority find them useful despite AI's insufficient accuracy at times and its inability to deal with sophisticated tasks. The developers report a variety of main pain points about programming while understanding code (especially at large scale and/or when written by others) stands out as the most prominent one. The developers also acknowledge not speaking English natively impacts their understanding of error messages or more specifically program analysis results to some extent. However, as English is the defacto language for programming, a skilled programmer is supposed to be reasonably proficient in English.

To answer RQ3.2, we present the developers' perceptions of future programming tool assistance. We found these developers are positive toward AI/ML, neutral (with a lean toward negative) toward eye tracking, and negative (polarized) toward gamification when asked about leveraging them to empower future programming tool assistance. The surveyed developers tend to believe that AI/ML (e.g., LLMs or Generative AI) is the next game changer in the field of programming.

5 Ethical Aspects

In this section, we address the ethical aspects of using eye-tracking and machine learning.

5.1 Ethical Aspects with Eye Tracking

Modern eye trackers usually employ near-infrared light for the illumination of target eyes. It is invisible to human eyes and normally deemed harmless (though strong infrared radiation can potentially cause hazards to human eyes in certain industry high-heat settings, e.g., glass making [22]). The sunlight [29] and many products we use in our household, e.g., certain home appliances, surveillance cameras, and wireless communication devices, emit infrared light. When conducting eye-tracking experiments with participants, its impact on one's health is negligible given the short exposure, usually within one hour. Additionally, many eye trackers are tested and certified for human safety by authorities from where they are manufactured. When eye tracking is integrated into a programming environment for potential long-lasting use, in our view, the tool makers should always give users the option to switch it off despite the harm being minimal.

Another concern with eye tracking raised by prospective users from our survey is about privacy. As introduced in the background section, contemporary eye trackers have developed to be physically unobtrusive. One will hardly notice it when an eye tracker attached to the desktop screen is in use. But whether users want their eye movement to be monitored and their gaze data to be stored and analyzed is another question. It is worth more discussion, especially in a work environment, about whether their gaze behavior will be accessible to their employers, supervisors, or peers. Different users may have different preferences, e.g., supervisors and seasoned programmers may not mind sharing theirs for didactic purposes whereas new employees or junior programmers may feel uncomfortable with that. Therefore, toolmakers should always request users' consent to enable eye tracking in a programming environment. The eye tracking functionality should be inactive by default unless users choose to activate it. Again, users should be informed about and given the option to opt out anytime and unconditionally. Ideally, the options and consent degree should be of great granularity, e.g., to what extent the gaze data will be collected and stored, whether it is shared, and with whom it will be shared. In the case that data will be aggregated for use, anonymization of specific users should be implemented.

5.2 Ethical Aspects with Machine Learning

We discuss the ethical aspect concerning the use of ML planned in our future studies. According to the meta-analysis study by Jobin et al. [47], the high-level principles of AI ethics are concerned with transparency, justice, fairness and equity, non-maleficence, responsibility, and privacy.

- **Transparency.** For transparency, we intend to prioritize the use of more explainable ML models when the performances are comparable and conduct feature ranking to understand the outstanding contributors. We are also open to publishing our code.
- **Justice and Fairness.** Justice and fairness are about ensuring the diversity of the data used especially for training the models. As depicted in Paper II, we are aware of the bias that can be introduced by the lack of diversity and balance in a data set. We plan to utilize tailored data argumentation [99], synthetic data generation [35], and pre-trained models to mitigate this.
- **Equity.** Equity can be translated into accessibility described by Siau and Wang [84]. The eye-tracking technology itself is in general accessible and particularly helpful to handicapped people. Our tool assistance intends to cater to not just conventional programmers or professional developers but the bigger and more diverse population in terms of their programming expertise, computer literacy, educational background, age, and language preferences and proficiency.

- **Non-maleficence.** In plain words, non-maleficence is not to do harm. This concerns safety and security, both on the personal level and on the holistic level. To the programmer individual, by design, the ML-enabled feature is deactivated. We aim to hand over to the programmer a high degree of controllability over the feature. On a holistic level, AI is perceived to cause job loss or replacement [84]. Our goal is to assist programmers as a whole to do their job better and particularly empower vernacular programmers who may need more support, but not to replace them.
- **Responsibility.** Responsibility means accountability, which requires us as AI developers to prepare the data and train the ML models responsibly. We will focus on pinpointing potential biases to prevent harm to the users during the development of the ML models.
- **Privacy.** For privacy, we will prevent the involuntary collection and use of one's gaze data. We also plan to reuse the open data set curated by the research community. Moreover, we intend to treat privacy concerns with synthetic data generation [35] and privacy-preserving federated learning [105].

We further share the point made by Mittelstadt [59]: "Principles alone cannot guarantee ethical AI". It is a complex and deep topic debated heatedly as it is compounded by many factors such as business competition, country race, and individual awareness and commitment [40]. Pursuing ethics as a process and shifting ethics to organizations may be some of the directions to drive this forward as Mittelstadt suggested. Nevertheless, we as AI developers commit to practicing in accordance with the abovementioned ethical principles to serve our fiduciary duties.

6 Threats to Validity

In this section, we address the threats to validity for the three studies we have undertaken.

6.1 Threats to Validity for Paper I and Paper II

For mapping studies, the key threats to validity reside in achieving completeness (maximizing the recall) and rejecting primary studies falsely (minimizing false negatives) [12]. To achieve completeness, we devised our search strings carefully and chose representative platforms for search. We examined and refined our search strings to achieve exhaustive hits, complimenting them with informal snowballing and inspection of publications from influential venues in the study domain. To ensure the quality of selection we either applied pilot studies on a good percentage of the papers retrieved to formally examine the inter-reliability among the authors (for Paper I) or discussed each one of the papers included (for Paper II). Another

significant threat to the validity of mapping studies is the quality of data extraction and synthesis. We are confident in this aspect as we had very high inter-reliability and discussed extensively both pre- and post-extraction to reach a consensus for eliminating misconceptions and reporting trends or topics identified.

6.2 Threats to Validity for Paper III

For surveys, sampling bias poses a common threat to the representativeness of the population selected and then to their conclusion validity or generalizability. Since we adopted convenience sampling, our samples skew toward developers who reside in Sweden and China. We tried to mitigate this by advertising our survey through a wide range of channels. However, location is not the primary attribute we look into for developers in this study. Additionally, some of the findings from the survey, e.g., some reasons why developers do not use program analysis, align with what we see in the literature [72, 73]. Finally, this survey aimed to understand practitioners' experience with and perception of programming tool assistance, hence, the findings do not apply to another distinct group of programmers, e.g., novices who may be students or learners.

7 Conclusions and Future Work

In conclusion, we have conducted three studies to answer the research question: How can eye tracking support programmers? Until this point, we have not fully addressed this overarching RQ. This is in part because we are investigating a "non-routine problem" [95], which implies possibly there is no fixed or pre-established solution. Nonetheless, the three studies we have conducted thus far help pave the way for a promising answer. The first mapping study constitutes the problem constructs which help us conceptualize the general problems. The second mapping study provides us with mostly solution design constructs and partly problem constructs again. It helps us paint out skeleton solution designs while again revealing the problems tied to these solutions that need to be further addressed. The third survey paper constitutes mainly problem instances and in part solution design constructs. It unearths a series of first-hand concrete problems that we can attempt to solve in a way of "problem setting" [77]. In the meantime, it also gathers evidence for us to select from multiple available backbone technologies for the solution design. These three studies together have built knowledge in us to work toward finding the best way to introduce gaze-assisted support to programmers.

7.1 Future Work

For future work, we contemplate plans as follows:

- **Participatory Design Study.** Practice participatory design with programmers to weed out design options and find viable designs that are worth implementation. Traditionally, programming languages and developer tools (e.g., formal specification [61]) were designed by experts or a small group of highly specialized programmers with little involvement of vernacular programmers at the early stage despite their constituting the greatest portion of the user population of these languages and tools [83]. We thus believe it is suitable to apply participatory design theory [14] in our exploratory research for designing gaze-based tool assistance.
- **Machine Learning with Gaze Data Study.** Explore the open gaze dataset curated by the research community with convolutional neural network models and transformer variations. There is a need in this respect since the artifacts and data sets of most related work are not available and thus cannot be reused or replicated. Since the raw data outputted by an eye-tracker is time series data, we believe it will be beneficial for us to experiment with different ML techniques specific to time series data. For instance, we believe some prominent emotional states of a programmer such as frustration and confusion may be solved by the anomaly detection task associated with time series data based ML [17]. We envision such anomalies can be used to "alarm" the tool to trigger more tailored assistance for the programmer. Additionally, in the scenario of aggregating programmers' gaze data, leveraging privacy-preserving federated learning [105] may reduce programmers' privacy concerns.
- **Proof of Concept and Evaluation Study.** Develop a proof of concept system with eye-tracking integrated and evaluate it with programmers iteratively. This will help us gather user feedback and incorporate their inputs into refining the tool assistance.

REFERENCES

[1] Amine Abbad-Andaloussi, Thierry Sorg, and Barbara Weber. Estimating developers' cognitive load at a fine-grained level using eye-tracking measures. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 111–121, 2022.

[2] Maike Ahrens, Kurt Schneider, and Melanie Busch. Attention in software maintenance: An eye tracking study. In *Int. Workshop on Eye Movements in Programming (EMIP)*, page 2 – 9, 2019.

[3] Zubair Ahsan and Unaizah Obaidellah. Predicting expertise among novice programmers with prior knowledge on programming tasks. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1008–1016. IEEE, 2020.

[4] Cognition AI. SWE-bench technical report, 3 2024. Retrieved 4 Apr, 2024 from `https://www.cognition-labs.com/post/swe-bench-technical-report`.

[5] Naser Al Madi, Drew Guarnera, Bonita Sharif, and Jonathan Maletic. Emip toolkit: A python library for customized post-processing of the eye movements in programming dataset. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–6, 2021.

[6] Fayiq Alghamdi. Why do female students choose to study cs in the kingdom of saudi arabia? In *2017 International Conference on Learning and Teaching in Computing and Engineering (LaTICE)*, pages 49–53. IEEE, 2017.

[7] Salwa Aljehane, Bonita Sharif, and Jonathan Maletic. Determining differences in reading behavior between experts and novices by investigating eye movement on source code constructs during a bug fixing task. *Eye Tracking Research and Applications Sym. (ETRA)*, 2021.

[8] Apple. The Home View on Apple Vision Pro, 10 2023. Retrieved Apr 3, 2024 from `https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/`.

[9] Google Arts and Culture. Women in Computing: a British Perspective. Retrived 3 Apr, 2024 from `https://artsandculture.google.com/story/QAVBBiKo-gMA8A`.

[10] Andrew Begel and Hana Vrzakova. Eye movements in code review. In *Workshop on Eye Movements in Programming (EMIP)*, 2018.

[11] Ian Bertram, Jack Hong, Yu Huang, Westley Weimer, and Zohreh Sharafi. Trustworthiness perceptions in code review: An eye-tracking study. In *Int. Sym. on Empirical Software Engineering and Measurement*, 2020.

[12] David Budgen, Pearl Brereton, Sarah Drummond, and Nikki Williams. Reporting systematic reviews: Some lessons from a tertiary study. *Information and Software Technology*, 95:62–74, 2018.

[13] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha E. Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: relaxing the linear order. In *Int. Conf. on Program Comprehension (ICPC)*, pages 255–265, 2015.

[14] Susanne Bødker, Christian Dindler, and Ole Sejer Iversen. *Participatory design*. Springer Nature, 5 2022.

[15] Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. An empirical study on the efficiency of different design pattern representations in uml class diagrams. *Empirical Softw. Engg.*, 15(5):493–522, oct 2010.

[16] Shiwei Cheng, Jialing Wang, Xiaoquan Shen, Yijian Chen, and Anind Dey. Collaborative eye tracking based code review through real-time shared gaze visualization. *Frontiers of Computer Science*, 16, 06 2022.

[17] Kukjin Choi, Jihun Yi, Changhwa Park, and Sungroh Yoon. Deep learning for anomaly detection in time-series data: Review, analysis, and guidelines. *IEEE access*, 9:120043–120065, 2021.

[18] Wikipedia contributors. Programming language, 11 2001. Retrieved Apr 3, 2024 from `https://en.wikipedia.org/wiki/Programming_language`.

[19] Wikipedia contributors. Front panel, 12 2023. Retrieved Apr 3, 2024 from `https://en.wikipedia.org/wiki/Front_panel`.

[20] Wikipedia contributors. Ballerina (programming language), 1 2024. Retrieved Apr 3, 2024 from `https://en.wikipedia.org/wiki/Ballerina_(programming_language)`.

[21] Wikipedia contributors. Computer, 3 2024. Retrieved Apr 3, 2024 from `https://en.wikipedia.org/wiki/Computer`.

[22] Wikipedia contributors. Infrared, 3 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Infrared.

[23] Wikipedia contributors. Internet, 3 2024. Retrieved from 3 Apr, 2024 https://en.wikipedia.org/wiki/Internet.

[24] Wikipedia contributors. Personal computer, 2 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Personal_computer.

[25] Wikipedia contributors. Programmer, 3 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Programmer.

[26] Wikipedia contributors. Punched card, 3 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Punched_card.

[27] Wikipedia contributors. Scratch (programming language), 3 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Scratch_(programming_language).

[28] Wikipedia contributors. Software crisis, 1 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Software_crisis.

[29] Wikipedia contributors. Sunlight, 3 2024. Retrieved Apr 3, 2024 from https://en.wikipedia.org/wiki/Sunlight.

[30] Wikipedia contributors. Women in computing, 4 2024. Retrived 3 Apr, 2024 from https://en.wikipedia.org/wiki/Women_in_computing.

[31] Martha Crosby, Jean Scholtz, and Susan Wiedenbeck. The roles beacons play in comprehension for novice and expert programmers. 07 2002.

[32] Martha E. Crosby and Jan Stelovsky. How do we read algorithms? a case study. *Computer*, 23:25–35, 1990.

[33] Andrew Duchowski and Andrew Duchowski. Eye tracking techniques. *Eye tracking methodology: Theory and practice*, pages 51–59, 2007.

[34] Emelie Engström, Margaret-Anne Storey, Per Runeson, Martin Höst, and Maria Teresa Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25:2630–2660, 2020.

[35] Alvaro Figueira and Bruno Vaz. Survey on synthetic data generation, evaluation methods and gans. *Mathematics*, 10(15):2733, 2022.

[36] World Economic Forum. Here's how technology can help close Nigeria's gender gap, 2 2023. Retrived 3 Apr, 2024 from `https://www.weforum.org/agenda/2023/02/how-technology-can-help-close-nigeria-gender-gap/`.

[37] Carol Frieze and Jeria L Quesenberry. *Cracking the digital ceiling: Women in computing around the world*. Cambridge University Press, 2019.

[38] Thomas Fritz, Andrew Begel, Sebastian C Müller, Serap Yigit-Elliott, and Manuela Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th international conference on software engineering*, pages 402–413, 2014.

[39] Hartmut Glücker, Felix Raab, Florian Echtler, and Christian Wolff. Eyede: gaze-enhanced software development environments. CHI EA '14, page 1555–1560, New York, NY, USA, 2014. Association for Computing Machinery.

[40] Thilo Hagendorff. The ethics of ai ethics: An evaluation of guidelines. *Minds and machines*, 30(1):99–120, 2020.

[41] Hiroto Harada and Minoru Nakayama. Estimation of reading ability of program codes using features of eye movements. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–5, 2021.

[42] Mar Hicks. *Programmed inequality: How Britain discarded women technologists and lost its edge in computing*. MIT press, 2017.

[43] Haytham Hijazi, José Cruz, João Castelhano, Ricardo Couceiro, Miguel Castelo-Branco, Paulo de Carvalho, and Henrique Madeira. ireview: an intelligent code review evaluation tool using biofeedback. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 476–485. IEEE, 2021.

[44] Haytham Hijazi, José Cruz, João Castelhano, Ricardo Couceiro, Miguel Castelo-Branco, Paulo de Carvalho, and Henrique Madeira. ireview: an intelligent code review evaluation tool using biofeedback. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 476–485, 2021.

[45] Kenneth Holmqvist, Richard Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van De Weijer. *Eye tracking: A comprehensive guide to methods, paradigms, and measures*. Oxford University Press, first edition, 2011.

[46] Yu Huang, Kevin Leach, Zohreh Sharafi, Nicholas McKay, Tyler Santander, and Westley Weimer. Biases and differences in code review using medical

imaging and eye-tracking: Genders, humans, and machines. In *European Software Engineering Conf. and Sym. on the Foundations of Software Engineering (ESEC/FSE)*, page 456 – 468, 2020.

[47] Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of ai ethics guidelines. *Nature machine intelligence*, 1(9):389–399, 2019.

[48] Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. Applying machine learning to gaze data in software development: a mapping study. In *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications*, ETRA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[49] Peng Kuang, Emma Söderberg, and Martin Höst. Developers' perspective on today's and tomorrow's programming tool assistance: A survey. In *Proceedings of the 2024 International Conference on the Art, Science, and Engineering of Programming*, 4 2024.

[50] Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. Toward gaze-assisted developer tools. In *2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 49–54, 2023.

[51] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. Mining biometric data to predict programmer expertise and task difficulty. *Cluster Computing*, 21:1097–1107, 2018.

[52] Jenny T Liang, Chenyang Yang, and Brad A Myers. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE, 2024.

[53] Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. Guidelines for conducting surveys in software engineering. 2015.

[54] Chao Liu, Xuanlin Bao, Hongyu Zhang, Neng Zhang, Haibo Hu, Xiaohong Zhang, and Meng Yan. Improving chatgpt prompt for code generation, 2023.

[55] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. Experiences from using code explanations generated by large language models in a web software development e-book, 2022.

[56] A. T. McCabe, M. Björkman, J. Engström, P. Kuang, E. Söderberg, and L. Church. Ironies of programming automation: Exploring the experience

of code synthesis via large language models. In *Proceedings of the 2024 International Conference on the Art, Science, and Engineering of Programming*, 2024.

[57] Roberto Minelli, Andrea Mocci, and Michele Lanza. I know what you did last summer - an investigation of how developers spend their time. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 25–35, 2015.

[58] Ieva Miseviciute. How do eye trackers work? — A tech-savvy walk-through. Retrieved Apr 3, 2024 from `https://www.tobii.com/resource-center/learn-articles/how-do-eye-trackers-work`.

[59] Brent Mittelstadt. Principles alone cannot guarantee ethical ai. *Nature machine intelligence*, 1(11):501–507, 2019.

[60] United Nations. Increased women's, girls' participation in digital technology crucial to economies, global sustainability, speakers tell Commission, as session continues | Meetings coverage and press releases, 3 2023. Retrived 3 Apr, 2024 from `https://press.un.org/en/2023/wom2224.doc.htm`.

[61] Peter Naur. *Computing: A human activity*. ACM, 1992.

[62] Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), 2018.

[63] Australian Bureau of Statistics. 2021 Census data shows Australia going high tech, 10 2022. Retrived 3 Apr, 2024 from `https://www.abs.gov.au/media-centre/media-releases/2021-census-data-shows-australia-going-high-tech`.

[64] OpenAI. Introducing ChatGPT, 11 2022. Retrieved Apr 3, 2024 from `https://openai.com/blog/chatgpt`.

[65] Mazliza Othman and Rodziah Latih. How the perception of young malaysians toward science and mathematics influences their decision to study computer science. *Cracking the Digital Ceiling: Women in Computing Around the World*, pages 276–282, 2019.

[66] Bharat Paudyal, Chris Creed, Maite Frutos-Pascual, and Ian Williams. Voice-eye: A multimodal inclusive development environment. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, page 21–33, New York, NY, USA, 2020. Association for Computing Machinery.

[67] Norman Peitek, Janet Siegmund, and Sven Apel. What drives the reading order of programmers? an eye tracking study. In *Int. Conf. on Program Comprehension (ICPC)*, page 342–353, 2020.

[68] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[69] Rachel Potvin and Josh Levenberg. Why google stores billions of lines of code in a single repository. *Commun. ACM*, 59(7):78–87, jun 2016.

[70] Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. Eyenav: Gaze-based code navigation. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*, NordiCHI '16, New York, NY, USA, 2016. Association for Computing Machinery.

[71] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. *Contemporary empirical methods in software engineering*, pages 127–147, 2020.

[72] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at google. *Commun. ACM*, 61(4):58–66, mar 2018.

[73] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Soderberg, and Collin Winter. Tricorder: Building a program analysis ecosystem. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 598–608, 2015.

[74] André L. Santos. Javardeye: Gaze input for cursor control in a structured editor. In *Companion Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming*, Programming '21, page 31–35, New York, NY, USA, 2021. Association for Computing Machinery.

[75] William Saranpää, Felix Apell Skjutar, Johan Heander, Emma Söderberg, Diederick C. Niehorster, Olivia Mattsson, Hedda Klintskog, and Luke Church. Gander: a platform for exploration of gaze-driven assistance in code review. In *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications*, ETRA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[76] Advait Sarkar, Andrew D. Gordon, Carina Negreanu, Christian Poelitz, Sruti Srinivasa Ragavan, and Ben Zorn. What is it like to program with artificial intelligence?, 2022.

[77] Donald A. Schön. *The reflective practitioner*. Routledge, 1 1983.

[78] Robert W. Sebesta. *Concepts of Programming Languages*. Pearson Education Limited, 11 edition, 1 2019.

[79] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. itrace: enabling eye tracking on software artifacts within the ide to support software engineering tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 954–957, New York, NY, USA, 2015. Association for Computing Machinery.

[80] Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. Toward an objective measure of developers' cognitive activities. *Transactions on Software Engineering and Methodology*, 30(3), 2021.

[81] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25:3128–3174, 2020.

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

[83] Mary Shaw. Myths and mythconceptions: what does it mean to be a programming language, anyhow? *Proc. ACM Program. Lang.*, 4(HOPL), apr 2022.

[84] Keng Siau and Weiyu Wang. Artificial intelligence (ai) ethics: ethics of ai and ethical ai. *Journal of Database Management (JDM)*, 31(2):74–87, 2020.

[85] Herbert A. Simon. *The Sciences of the artificial*. 1 1969.

[86] Statistics Sweden. Employees 16-64 years at national level by occupation (3-digit SSYK 2012), industry SNI2007 (aggr. level), age and sex. Year 2019 - 2021.

[87] Talon. Talon: Powerful Hands-free Input. Retrieved 3 Apr, 2024 from `https://talonvoice.com/`.

[88] Divy Thakkar, Nithya Sambasivan, Purva Kulkarni, Pratap Kalenahalli Sudarshan, and Kentaro Toyama. The unexpected entry and exodus of women in computing and hci in india. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[89] ThisDay. 2022: Nigerian women in tech break new grounds, 1 2022. Retrived 3 Apr, 2024 from `https:`

//www.thisdaylive.com/index.php/2022/01/31/2022-nigerian-women-in-tech-break-new-grounds.

[90] Haoye Tian, Weiqi Lu, Tsz On Li, Xunzhu Tang, Shing-Chi Cheung, Jacques Klein, and Tegawendé F. Bissyandé. Is chatgpt the ultimate programming assistant – how far is it?, 2023.

[91] Tobii. Eye tracking fully integrated and baked right into the very latest high performance gaming devices from alienware, acer and msi. https://gaming.tobii.com/products/laptops/ (Oct 3, 2022).

[92] Niilo V Valtakari, Ignace TC Hooge, Charlotte Viktorsson, Pär Nyström, Terje Falck-Ytter, and Roy S Hessels. Eye tracking in human interaction: Possibilities and limitations. *Behavior Research Methods*, pages 1–17, 2021.

[93] Joan E van Aken. Management research based on the paradigm of the design sciences: the quest for field-tested and grounded technological rules. *Journal of management studies*, 41(2):219–246, 2004.

[94] Maureen Villamor and Ma Mercedes Rodrigo. Predicting successful collaboration in a pair programming eye tracking experiment. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, pages 263–268, 2018.

[95] Willemien Visser. Designing as construction of representations: A dynamic viewpoint in cognitive design research. *Human–Computer Interaction*, 21(1):103–152, 2006.

[96] Hana Vrzakova, Andrew Begel, Lauri Mehtätalo, and Roman Bednarik. Affect recognition in code review: An in-situ biometric study of reviewer's affect. *Journal of Systems and Software*, 159:110434, 2020.

[97] Braden Walters, Michael Falcone, Alexander Shibble, and Bonita Sharif. Towards an eye-tracking enabled ide for software traceability tasks. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 51–54, 2013.

[98] Chun-Chia Wang, Jason C. Hung, Shih-Cheng Wang, and Yueh-Min Huang. Visual attention analysis during program debugging using virtual reality eye tracker. *Lecture Notes in Computer Science*, 11937:97 – 106, 2019.

[99] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[100] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C. Schmidt. A prompt pattern catalog to enhance prompt engineering with chatgpt, 2023.

[101] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design, 2023.

[102] Shengqiong Wu, Hao Fei, Leigang Qu, Wei Ji, and Tat-Seng Chua. Next-gpt: Any-to-any multimodal llm. *arXiv preprint arXiv:2309.05519*, 2023.

[103] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring program comprehension: A large-scale field study with professionals. *IEEE Transactions on Software Engineering*, 44(10):951–976, 2018.

[104] John Yang, Carlos E. Jimenez, Alexander Wettig, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent computer interfaces enable software engineering language models, 2024.

[105] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. 10(2), jan 2019.

[106] Guillermina Yansen and Mariano Zukerfeld. Why don't women program? exploring links between gender, technology and software. *Science, Technology and Society*, 19(3):305–329, 2014.

[107] Shehnaaz Yusuf, Huzefa Kagdi, and Jonathan I. Maletic. Assessing the comprehension of uml class diagrams via eye tracking. In *15th IEEE International Conference on Program Comprehension (ICPC '07)*, pages 113–122, 2007.

[108] J.D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qian Yang. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[109] Li Zhang, Jianxin Sun, Cole Peterson, Bonita Sharif, and Hongfeng Yu. Exploring eye tracking data on source code via dual space analysis. In *2019 Working Conference on Software Visualization (VISSOFT)*, pages 67–77. IEEE, 2019.

[110] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement, 2024.

INCLUDED PAPERS

PAPER I

TOWARD GAZE-ASSISTED DEVELOPER TOOLS

Abstract

Many crucial activities in software development are linked to gaze and can potentially benefit from gaze-assisted developer tools. However, despite the maturity of eye trackers and the potential for such tools, we see very few studies of practitioners. Here, we present a systematic mapping study to examine recent developments in the field with a focus on the experimental setup of eye-tracking studies in software engineering research. We identify two gaps regarding studies of practitioners in realistic settings and three challenges in existing experimental setups. We present six recommendations for how to steer the research community toward gaze-assisted developer tools that can benefit practitioners.

1 Introduction

One of the main activities of a software developer is reading code [25, 45], an activity driven by gaze. Still, research using eye trackers to study the gaze behavior of software developers during their day-to-day work is not common [27, 36]. Gaze-driven tools become even more attractive when considering the increase in remote work and online collaboration [46]. There are a number of activities in software engineering where we believe that gaze analysis and eye-tracking data could contribute to research, e.g., in cooperative work, both in the same location and geographically distributed, and in code comprehension.

Eye trackers used to be expensive and required trained personnel to operate, effectively confining their use to lab environments. But this is not the case anymore. In the last decades, rapid technical development has greatly increased the feasibility of in-situ acquisition of eye-tracking data, both by the availability of cheap dedicated eye trackers and eye tracking via commodity webcams. It is reasonable

to assume eye trackers will soon be ubiquitous at developers' desks, integrated into monitors, laptops, and cell phones [12, 41, 42].

Considering research in software engineering, the possibility of using eye tracking in software development has been identified [37, 39] and we see an increase in studies that utilize eye trackers to understand e.g., program comprehension [27, 36]. Also, it can be utilized as a means for data collection in empirical software engineering research [38]. However, when considering the experimental setup of eye-tracking studies so far, we see a dominance of lab studies, with students and educational material [27]. We see very few studies considering the day-to-day activities performed in professional software engineering settings. Yet, there is evidence that experts and novices have different gaze patterns [7, 33]. How well can we build tools for practitioners with gaze data from students in an educational lab setting?

In this paper, we systematically review recent studies utilizing eye trackers in software engineering research to investigate recent trends (Section 2). We find a persistent lack of eye-tracking experiments conducted outside of the lab environment and a dominance of student participants (Section 3). With these results in mind, we discuss challenges in utilizing results and possible paths forward (Section 4).

2 Method

To develop a thorough overview of current use of eye-tracking in software engineering research and to address the question: *how is eye-tracking used in software engineering research?*, we conducted a mapping study [29] summarized in Table 1. We explain the steps and the limitations of the study below. Supplementary material with our data set is available online[1].

Mapping Study Steps In step 1, we conducted a search in Scopus[2], an "abstract and citation database" with references to articles from journals and conferences from established publishers (e.g., Elsevier, Springer, and IEEE). Scopus was chosen as a source because we deemed it to provide a representative view of available literature in the studied research area. We used the following search string (with line numbers added):

```
SUBJAREA ( comp ) AND                                                  1
TITLE-ABS-KEY (                                                        2
   (  {debugging} OR {programming} OR                                  3
      {source code navigation} OR                                      4
      {code browsing} OR {code search} OR                              5
      {code review} ) AND                                              6
   (  {eye tracking} OR {eyetracking} OR                               7
      {gaze} OR {eye movement} OR                                      8
```

[1] https://portal.research.lu.se/files/137278187/ICSE_NIER_2023_KuangEtAl_artifact.xlsx

[2] https://www.scopus.com

Table 1: Summary of steps in mapping study

	Steps
1.	Search in academic database, resulting in 509 papers.
2.	All authors review 10% of papers (titles and abstracts), with Kappa showing strong inter-rater agreement.
3.	Updated search in academic database, resulting in 513 papers.
4.	First author reviews all papers (titles and abstracts), resulting in 204 papers.
5.	All authors review 10% of papers (full content) in pairs and agree to keep all papers.
6.	All authors develop a data collection scheme together.
7.	Joint decision to focus on papers from 2018 and later, resulting in 136 papers.
8.	First author reviews remaining papers (full content), resulting in 86 papers.
9.	First author extracts data according to the data collection scheme, resulting in 71 papers.
10.	Joint analysis of data set and extraction of summaries by the first and second author.

```
      {gaze estimation} ) ) AND                                    9
( EXCLUDE ( DOCTYPE , "cr" ) )                                     10
```

The first line focuses on the search in the area of computer science and the last line removes irrelevant documents (e.g., book reviews). Line 2 states that we search in titles, keywords, and abstracts, and the two main areas we require in each article are stated on lines 2–6 (programming) and 7–9 (eye tracking). The search resulted in 509 papers (April 22, 2022).

In step 2, we conducted a pilot review of the titles and abstracts of the first 10% of the found papers (51 papers) to develop an approach for sorting out irrelevant papers based on only titles and abstracts. We selected the 51 most recent papers to ensure this approach is based on the most recent research. All authors then reviewed these separately. The inclusion criteria we applied were: First, the stimuli must be code (studies solely with pseudocode stimuli were excluded); Second, the study must contain experiments with humans (but studies with children were excluded); Third, the study must be generating some data rather than entirely reusing preexisting data. The exclusion criteria we employed were: the paper is not in English, peer-reviewed, or available in full text. We then performed a Light's Kappa analysis of the inter-rater reliability of multiple raters, and got a Kappa score of 0.86, indicating that the magnitude of agreement is "strong" [9]. With the strong agreement, one author could proceed to review the rest of the papers.

Next, we re-ran the search (step 3, on 27 June 2022) since some time had

passed, and this resulted in 4 additional papers, i.e., 513 papers in total. Then the first author proceeded with selecting the relevant papers (step 4), ending up with 204 potentially relevant papers.

In step 5, we selected the first 20 of the included papers (approximately 10%) to conduct a quality assurance study with the full text of each paper. We assigned two co-authors to cross-review each paper. We had six unique pairs which covers 18 of the 20 papers by repeating three times. For the remaining two papers, we deliberately assigned them to the pairs who had a relatively low agreement in the prior pilot review. After this process, we discussed our assessment of the quality of the papers we read. We unanimously agreed on the inclusion of 17 papers. For the remaining three papers, we used a majority voting mechanism to decide on the inclusion of two papers; we also included a paper when there was a tie. After review, all 20 examined papers were included. Since we had an agreement on which papers to remove and which to keep, we concluded that one author could review the rest of the papers.

In steps 6-9, we designed a data collection form (step 6) and used it to conduct pilot data extraction on the aforementioned papers. The first author also recorded a time estimate for reading each paper. We discussed and revised the form after this procedure. Based on review time and the existence of a previous study covering literature up until 2017 [27], we decided to limit the investigation to papers from 2018 and later (step 7). After that the first author reviewed the remainder of the papers (step 8), resulting in 86 papers, and also extracted data from the papers (step 9).

In the final step, we coded details about the experimental setup (the participants, stimuli/artifacts, devices, environment, and methods). We only considered distinct primary studies. That is, publications on the same data set were counted only once and it was the first or original study we considered. In total, we excluded 15 papers with this criterion, leaving 71 papers to be included in the reported results.

Limitations There are some limitations and threats to validity in this type of research (e.g., [6]). Even if a structured process for selecting papers is applied there is a risk, e.g., that some papers are rejected falsely. We have carefully reviewed papers and followed a process where we could reach a consensus about papers, which we believe increased the quality of the selection.

Also, it is not possible to get complete coverage of primary studies. However, we selected a well-known database and carefully designed the search string, which we believe increased the completeness. If very few primary studies were identified that could be seen as an indication that the selection was too narrow. However, in this case, we identify not so few relevant sources (in the magnitude of 15 per year). We compare our results to trends, but it is impossible to get a complete list of trends in the broader area. We base our analysis on our experience from research in the fields and we believe this experience to be sufficient to identify future paths. To not miss important perspectives, we spent effort on identifying and obtaining

consensus in the review process.

3 Results

Fig. 1(a) shows the summary of roles of participants in the studies we found, split into students, practitioners, mixed, and unspecified. A study was labeled as having mixed participants when it included participants with more than one role, e.g., students and researchers [10,11]. The data show that while there was no change in the number of studies using only students during the years covered by our review, there was a notable increase in the number of studies using participants with more than one role in 2019–2021, compared to 2018. Across the years, the vast majority of studies (92.96%) used students. These studies used either only student participants (61.97%) or used students in combination with researchers (8.45%, e.g., [10,11]), practitioners (12.68%, e.g., [8,13,30]) or both (8.45%, e.g., [15,16]). Conversely, only 22.54% of studies included practitioners as participants.

Finding 1: The vast majority of studies (92.96%) used students in eye-tracking experiments, with 61.97% using only students. 22.54% of studies used practitioners.

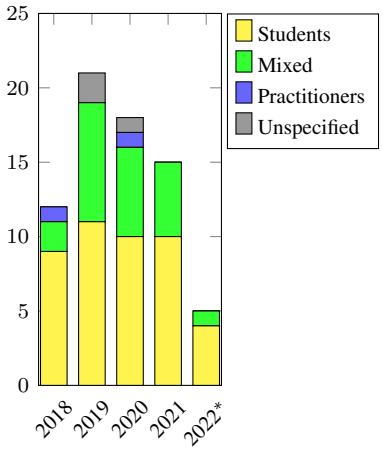
Fig. 1(b) shows in what context (setting & material) the reviewed studies were performed, split into 12 combinations of setting and material. The categories with the "mixed" prefix indicate studies that were conducted in more than one setting, usually involving different participant roles, e.g., Lab for Students and Workplace for Researchers/Practitioners [3,13].

We notice that laboratory studies using open-source software as the source of stimuli were more common in 2019 (e.g., [1,2]) and 2020 (e.g., [11,14,19]) than the other years. Only two studies were done in the workplace with closed-source software [5,43].

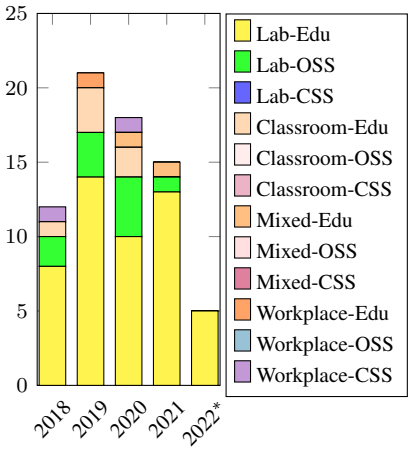
Finding 2: A majority of studies (84.51%) were conducted in a laboratory setting, either with educational materials (70.42%) or with materials adapted from open-source software (14.09%).

Fig. 1(c) shows what research methods were used by studies across the time period we reviewed. As the studies we examined all utilize eye tracking, the research method they adopted was either quantitative or mixed. Among these mixed studies, 68.75% (15.49% of all studies) used post-test/retrospective interviews to complement the quantitative analysis of eye movement data; the rest used either think-aloud, verbal Q&A, or a combination of the two.

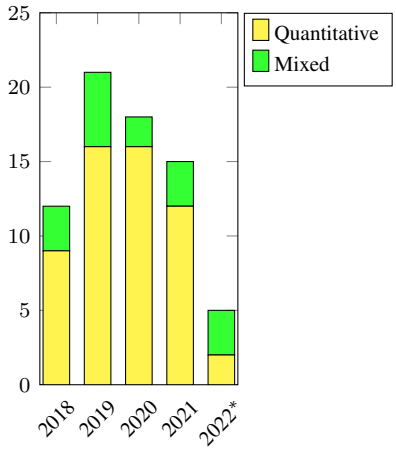
Moreover, ten distinct studies adopted other sensors/devices, e.g., fMRI (functional Magnetic Resonance Imaging, e.g., [19,34]), fNIRS (functional Near Infrared Spectroscopy, e.g., [14,34]), EEG (electroencephalogram, e.g., [21,22]) and HRV (heart rate variability, e.g., [16,23]), either simultaneously or subsequently



(a) Participant roles



(b) Study context



(c) Research method

Figure 1: Overview of experiment setup in selected papers split by year. The legends list coding categories for participant roles (a), study context (b) with a combination of location and material used, and study method (c). Note Edu=Educational, OSS=Open-Source Software, and CSS=Closed-source software. Mixed is used in all plots to indicate a combination of roles, contexts, or methods. *2022 includes research only up to June 27, 2022.

with eye trackers. Two studies used VR eye trackers instead of screen-based eye trackers to collect gaze data [20, 44].

Finding 3: A majority of studies (77.46%) used Quantitative research method while 22.54% adopted a Mixed approach.

4 Discussion

Our data combined with previous data [27], suggests a clear increase in research studies involving eye-tracking in the area of software engineering. We ended up with 71 papers in our study, which constitutes an average of 15.78 papers per year (71/4.5), while in the previous survey by Obaidellah et al. [27] they found 2.33 papers per year over the range 1990-2017, with 6.4 papers per year for the last 5 years in their data set (2012-2017). We see this as an encouraging development toward the utilization of gaze in developer tooling, but we also see gaps in the areas covered by this research and challenges in how to conduct research in this space.

Gap: Gaze behavior of practitioners. The study participants are predominately students (**Finding 1**), a finding that aligns with previous surveys of the field [27, 36]. Although the argument can be made that students in some cases can be seen as representatives of junior engineers [36], this still leaves a gap regarding the gaze behavior of more experienced practitioners.

Gap: Gaze behavior in realistic setups. Studies primarily take place in a lab or classroom setting, with mainly educational material and occasionally with material from open-source (**Finding 2**). While we acknowledge efforts to bring in more realism, with stimuli from open-source, and of course the few gems studying practitioners in the workplace (e.g., [43], [5]), there is generally little knowledge about gaze behavior in realistic software development settings. This aligns with previous surveys where *"most of the experiments reported in relation to programming were conducted in an institution for higher learning"* [27] (Section 3.4) with 75% of studies being carried out with students, teachers or researchers with a connection to programming courses and often with *"source code, texts, or models that can fit on one screen"* [36] (Section 5.5.3).

Challenge: Task incentive. Besides the realism of the setting and the stimuli, there is also something to be considered regarding the incentives of participants, which is not mentioned in earlier work [27, 36]. The incentive for doing a task may have an impact on the gaze behavior and thus create bias in findings. It is reasonable to assume that there could be a big gap between knowing that not finding a bug will have no severe consequences vs. knowing that not finding a bug (and fixing it) will affect thousands of users.

Challenge: Characterizing participant expertise. While it is plausible to divide participants into different groups and contrast them for studies, we observe the line researchers use to draw between novices and experts is not consistent in

the literature. While freshman students are typically treated as novices and practitioners as experts, senior and graduate students may be treated as intermediate, advanced, or novices. Variation in role assignment makes it difficult to compare and replicate studies. While previous surveys show that studies of differences between novices and experts are common [27, 36], they do not list this challenge.

Beyond the immediate challenges in the current practice in expertise characterization, the notion of expertise is by nature very difficult to measure and context-dependent. Expertise, proficiency, and task difficulty are intertwined and interact with each other. Expertise can vary between languages; an expert in C can be a novice in JavaScript. Past language experience matters; being a novice in your fifth language is not the same as the first time you learn to program [40]. Expertise may also vary within a language; solving the same problem for the fifth time is not the same as the first time. Lastly, expertise can also degrade over time; 20 years of experience in C may not be the same after spending 5 years developing in Python.

Challenge: Building cross-study knowledge. Eye tracking research deals with massive amounts of data [36] and a high degree of attention and rigor is required when using eye trackers to generate valid data. In addition, processing and analyzing eye movement data is also challenging. There has been a lack of guidance in the field [27,36], resulting in variation in experimental setups and problems with replication (e.g., [28], see also [17]). Variation in the meta-data of gathered data sets further makes it difficult to compare results and to combine results into larger data sets, to open up for automated pattern recognition via machine learning.

4.1 Toward gaze-assisted developer tools

With the presented gaps and challenges in mind, we present recommendations for how to drive the research in this area toward gaze-driven developer tools for practitioners.

Recommendation: Find paths to practitioners. While we acknowledge that it may be difficult for researchers to get access to practitioners [36], we still advocate for more studies with practitioners and preferably conducted in their natural work environment, a point also made by others [35]. We encourage researchers to take on the challenge of finding paths to practitioners, perhaps by considering other research methods or adaptions to their experimental setups. Are there ways to move to practitioners rather than moving practitioners to the lab? Are there ways to find a mutual benefit to their participation?

Recommendation: Let expertise be more complex. The current practice of characterizing expertise is to use years of programming, years of using a certain programming language, self-reported proficiency/confidence, and perceived task difficulty. While all these metrics are valid and relevant, we encourage researchers to consider alternative ways of characterizing expertise, capturing more of the nuance. Perhaps by capturing experience along more dimensions, for instance, past experience in different programming languages, and recent development activities.

Recommendation: Provide realistic incentives. In order to increase the validity it has been argued that not only expertise is important, but also the incentives for conducting tasks in experiments [18]. Incentives are probably clearer in an industrial setting, but incentives can be seen as orthogonal to subject experience, e.g. there may, to some extent, be clear incentives also in tasks that are part of student projects. We encourage researchers to obtain realistic incentives also with students as subjects.

Recommendation: Tee up for machine learning. As the field matures, practical guides for how to conduct eye-tracking experiments are emerging [35], as well as efforts to share data sets [4, 24]. High-quality data sets with gaze data open the door for machine learning and the incorporation of such techniques into developer tools for improved developer assistance. To enable this development, we encourage researchers to take on the challenge of creating larger data sets with gaze data from practitioners in realistic settings.

Recommendation: Characterize gaze in development. While shared gaze data sets enable the training of machine learning models, an important factor that makes such data sets useful lies in the richness of the meta-data that describe them (see also [26]). Describing the space where gaze plays a role in software development, is more complex than describing the space of an introductory programming course. While a course is simplified (e.g., typically focuses on one language and construction of small applications), software development is multi-faceted and complex (e.g., editing of multiple languages, remote collaboration with pair programming, code review, and whiteboard design discussions). We encourage researchers to take on the challenge of characterizing this "gaze space" in software development to enable knowledge building and gaze-driven tooling for software developers.

Recommendation: Involve participants in design. Software development is not only a cognitively-demanding task but can also be an emotion-draining task. Gaze data, along with other bio-sensors, open up possibilities to detect part of a developer's emotional state [35]. This data can potentially enrich the existing services of developer tools and also create new ones focused on emotional user experiences [31]. However, there is a risk that the cost of sharing bio data may outweigh the benefit of the assistance. We recommend researchers consider participatory design methods [32] in exploring such services.

5 Conclusions

To investigate how eye-tracking is used in software engineering research, we carried out a systematic mapping study. We focused on the experimental setups in the last 5 years and found that the majority of experiments are carried out with students, in a lab setting and with educational material as stimuli. We identify gaps in terms of studies with practitioners in realistic settings, and we identify

challenges in existing studies regarding task incentives, characterization of expertise, and variation in the practice around data gathering. To address these gaps and challenges, we present six recommendations aimed at steering the research community toward gaze-assisted developer tools useful for practitioners.

Acknowledgment

This work was partly funded by the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231), the Swedish Research Council (grant nbr. 2019-05658), and Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Nahla J. Abid, Bonita Sharif, Natalia Dragan, Hend Alrasheed, and Jonathan I. Maletic. Developer reading behavior while summarizing java methods: Size and context matters. In *Int. Conf. on Software Engineering (ICSE)*, page 384 – 395, 2019.

[2] Maike Ahrens, Kurt Schneider, and Melanie Busch. Attention in software maintenance: An eye tracking study. In *Int. Workshop on Eye Movements in Programming (EMIP)*, page 2 – 9, 2019.

[3] Salwa Aljehane, Bonita Sharif, and Jonathan Maletic. Determining differences in reading behavior between experts and novices by investigating eye movement on source code constructs during a bug fixing task. *Eye Tracking Research and Applications Sym. (ETRA)*, PartF169257, 2021.

[4] Roman Bednarik, Teresa Busjahn, Agostino Gibaldi, Alireza Ahadi, Maria Bielikova, ..., and Ian van der Linde. Emip: The eye movements in programming dataset. *Science of Computer Programming*, 198:102520, 2020.

[5] Andrew Begel and Hana Vrzakova. Eye movements in code review. In *Workshop on Eye Movements in Programming (EMIP)*, 2018.

[6] David Budgen, Pearl Brereton, Sarah Drummond, and Nikki Williams. Reporting systematic reviews: Some lessons from a tertiary study. *Information and Software Technology*, 95:62–74, 2018.

[7] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha E. Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. Eye movements in code reading: relaxing the linear order. In *Int. Conf. on Program Comprehension (ICPC)*, pages 255–265, 2015.

[8] Teresa Busjahn and Sascha Tamm. A Deeper Analysis of AOI Coverage in Code Reading. In *Eye Tracking Research and Applications Sym. (ETRA)*, 2021.

[9] A. J. Conger. Integration and generalization of kappas for multiple raters. *Psychological Bulletin*, 88, 1980.

[10] José Costa, Rohit Gheyi, Márcio Ribeiro, Sven Apel, Vander Alves, Baldoino Fonseca, Flávio Medeiros, and Alessandro Garcia. Evaluating refactorings for disciplining #ifdef annotations: An eye tracking study with novices. *Empirical Software Engineering*, 26, 2021.

[11] Benedito De Oliveira, Marcio Ribeiro, Jose Aldo Silva Da Costa, Rohit Gheyi, Guilherme Amaral, Rafael De Mello, Anderson Oliveira, Alessandro Garcia, Rodrigo Bonifacio, and Baldoino Fonseca. Atoms of Confusion: The Eyes Do Not Lie. *Int. Conf. Proc. Series*, page 243 – 252, 2020.

[12] EIZO. Eye Tracking Leader Tobii Pro Choose EIZOs Monitors for Behavioral Research Solution. https://www.eizoglobal.com/solutions/casestudies/tobii-pro/ (Oct, 2022).

[13] Selina N. Emhardt, Ellen M. Kok, Halszka Jarodzka, Saskia Brand-Gruwel, Christian Drumm, and Tamara van Gog. How experts adapt their gaze behavior when modeling a task to novices. *Cognitive Science*, 44(9), 2020.

[14] Sarah Fakhoury, Devjeet Roy, Yuzhan Ma, Venera Arnaoudova, and Olusola Adesope. Measuring the impact of lexical and structural inconsistencies on developers' cognitive load during bug localization. *Empirical Software Engineering*, 25(3):2140 – 2178, 2020.

[15] Florian Hauser, Stefan Schreistter, Rebecca Reuter, Jürgen Horst Mottok, Hans Gruber, Kenneth Holmqvist, and Nick Schorr. Code reviews in C++: Preliminary results from an eye tracking study. In *Eye Tracking Research and Applications Sym. (ETRA)*, 2020.

[16] Haytham Hijazi, José Cruz, João Castelhano, Ricardo Couceiro, Miguel Castelo-Branco, Paulo de Carvalho, and Henrique Madeira. ireview: an intelligent code review evaluation tool using biofeedback. In *Int. Sym. on Software Reliability Engineering (ISSRE)*, pages 476–485, 2021.

[17] K. Holmqvist, S. L. Örbom, I. T. C. Hooge, D. C. Niehorster, ..., and R. S. Hessels. Eye tracking: empirical foundations for a minimal reporting guideline. *Behavior Research Methods*, 2022.

[18] Martin Höst, Claes Wohlin, and Thomas Thelin. Experimental context classification: incentives and experience of subjects. In *Int. Conf. on Software Engineering (ICSE)*, pages 470–478, 2005.

[19] Yu Huang, Kevin Leach, Zohreh Sharafi, Nicholas McKay, Tyler Santander, and Westley Weimer. Biases and differences in code review using medical imaging and eye-tracking: Genders, humans, and machines. In *European*

Software Engineering Conf. and Sym. on the Foundations of Software Engineering (ESEC/FSE), page 456 – 468, 2020.

[20] Jason C. Hung and Chun-Chia Wang. The influence of cognitive styles and gender on visual behavior during program debugging: A virtual reality eye tracker study. *Human-centric Computing and Information Sciences*, 11, 2021.

[21] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. Mining biometric data to predict programmer expertise and task difficulty. *Cluster Computing*, 21(1):1097 – 1107, 2018.

[22] Yu-Tzu Lin, Yi-Zhi Liao, Xiao Hu, and Cheng-Chih Wu. Eeg activities during program comprehension: An exploration of cognition. *IEEE Access*, 9:120407 – 120421, 2021.

[23] Katerina Mangaroska, Kshitij Sharma, Dragan Gašević, and Michalis Giannakos. Multimodal learning analytics to inform learning design: Lessons learned from computing education. *Journal of Learning Analytics*, 7(3):79 – 97, 2020.

[24] Ian McChesney and Raymond Bond. Eye tracking analysis of code layout, crowding and dyslexia - an open data set. In *Sym. on Eye Tracking Research and Applications*, 2021.

[25] Roberto Minelli, Andrea Mocci, and Michele Lanza. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In *Int. Conf. on Program Comprehension (ICPC)*, pages 25–35, 2015.

[26] Diederick C. Niehorster, Michael Hildebrandt, Anthony Smoker, Halszka Jarodzka, and Nicklas Dahlströhm. Towards eye tracking as a support tool for pilot training and assessment. In *Int. Workshop on Eye-Tracking in Aviation (ETAVI)*, pages 17 – 28, 2020.

[27] Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), 2018.

[28] Norman Peitek, Janet Siegmund, and Sven Apel. What drives the reading order of programmers? an eye tracking study. In *Int. Conf. on Program Comprehension (ICPC)*, page 342–353, 2020.

[29] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[30] Cole S. Peterson. Investigating the effect of polyglot programming on developers. In *Sym. on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–2, 2021.

[31] Kathrin Pollmann, Mathias Vukelić, Niels Birbaumer, Matthias Peissner, Wilhelm Bauer, and Sunjung Kim. fnirs as a method to capture the emotional user experience: A feasibility study. In *Human-Computer Interaction. Novel User Experiences*, 2016.

[32] Toni Robertson and Jesper Simonsen. Challenges and Opportunities in Contemporary Participatory Design. *Design Issues*, 28(3):3–9, 2012.

[33] Jonathan A. Saddler, Cole S. Peterson, Patrick Peachock, and Bonita Sharif. Reading behavior and comprehension of C++ source code - a classroom study. In *Augmented Cognition*, 2019.

[34] Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. Toward an objective measure of developers' cognitive activities. *Transactions on Software Engineering and Methodology*, 30(3), 2021.

[35] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25:3128–3174, 2020.

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

[37] Bonita Sharif, Benjamin Clark, and Jonathan I. Maletic. Studying developer gaze to empower software engineering research and practice. In *Int. Sym. on Foundations of Software Engineering (FSE)*, page 940–943, 2016.

[38] Bonita Sharif and Niloofar Mansoor. Humans in empirical software engineering studies: An experience report. In *Int. Conf. on Software Analysis, Evolution and Reengineering (SANER)*, pages 1286–1292, 2022.

[39] Bonita Sharif and Timothy Shaffer. The use of eye tracking in software development. In Dylan D. Schmorrow and Cali M. Fidopiastis, editors, *Foundations of Augmented Cognition*, pages 807–816, 2015.

[40] Georg Simhandl, Philipp Paulweber, and Uwe Zdun. Design of an executable specification language using eye tracking. In *Workshop on Eye Movements in Programming (EMIP)*, pages 37–40, 2019.

[41] Tobii. Eye tracking fully integrated and baked right into the very latest high performance gaming devices from alienware, acer and msi. `https://gaming.tobii.com/products/laptops/` (Oct 3, 2022).

[42] Nachiappan Valliappan, Na Dai, Ethan Steinberg, Junfeng He, Kantwon Rogers, Venky Ramachandran, Pingmei Xu, Mina Shojaeizadeh, Li Guo, Kai Kohlhoff, and Vidhya Navalpakkam. Accelerating eye movement research via accurate and affordable smartphone eye tracking. *Nature Communications*, 11, 2020.

[43] Hana Vrzakova, Andrew Begel, Lauri Mehtätalo, and Roman Bednarik. Affect recognition in code review: An in-situ biometric study of reviewer's affect. *Journal of Systems and Software*, 159:110434, 2020.

[44] Chun-Chia Wang, Jason C. Hung, Shih-Cheng Wang, and Yueh-Min Huang. Visual attention analysis during program debugging using virtual reality eye tracker. *Lecture Notes in Computer Science*, 11937:97 – 106, 2019.

[45] Xin Xia, Lingfeng Bao, David Lo, Zhenchang Xing, Ahmed E. Hassan, and Shanping Li. Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *Transactions on Software Engineering*, 44(10), 2018.

[46] L. Yang, D. Holtz, S. Jaffe, S. Suri, S. Sinha, J. Weston, C. Joyce, N. Shah, K. Sherman, B. Hecht, and J. Teevan. The effects of remote work on collaboration among information workers. *Nature Human Behaviour*, 6:43–54, 2022.

PAPER II

APPLYING MACHINE LEARNING TO GAZE DATA IN SOFTWARE DEVELOPMENT: A MAPPING STUDY

Abstract

Eye tracking has been used as part of software engineering and computer science research for a long time, and during this time new techniques for machine learning (ML) have emerged. Some of those techniques are applicable to the analysis of eye-tracking data, and to some extent have been applied. However, there is no structured summary available on which ML techniques are used for analysis in different types of eye-tracking research studies.

In this paper, our objective is to summarize the research literature with respect to the application of ML techniques to gaze data in the field of software engineering. To this end, we have conducted a systematic mapping study, where research articles are identified through a search in academic databases and analyzed qualitatively. After identifying 10 relevant articles, we found that the most common software development activity studied so far with eye-tracking and ML is program comprehension, and Support Vector Machines and Decision Trees are the most commonly used ML techniques. We further report on limitations and challenges reported in the literature and opportunities for future work.

1 Introduction

Both eye tracking (ET) and machine learning (ML) technology have matured a lot in the last decade; eye tracking has reached a point where it is being integrated into

consumer laptops[1], and ML has seen significant progress (e.g., AlphaGo [19] and ChatGPT [21]). Programming can benefit from these two developments since eye trackers generate a lot of multifaceted data that may reveal a wealth of information about the user and their intentions, while machine learning is needed to distill these high dimensional data into actionable information [5]. Specifically, there is an opportunity to utilize ML techniques to both understand programmers' gaze behavior and to explore gaze-assisted developer tools [10].

However, despite the initial development of gaze data sets intended to accelerate research in this area [3, 4], there are limited guidelines for how to apply machine learning techniques to this data. Existing guidelines for employing eye tracking for studies in the field of software engineering focus on data gathering and metrics [17], but have little to say about making the collected data useful for the training and evaluation of machine learning models. Existing surveys of eye-tracking research studies in software engineering [10, 14, 18] give overviews with a focus on data gathering, experimental setups, and metrics [18], but likewise do not discuss the use of machine learning techniques in these studies, nor whether data gathered in these studies is suitable for the development of machine learning models. Likewise, more general guidelines for eye tracking studies [**?**, e.g.]]holmqvist2022guidelines do not pay particular attention to machine learning approaching in gaze data analysis.

In this paper, we aim to contribute to filling this gap by providing an overview of how machine learning techniques have been applied to gaze data so far in the software engineering literature. To this end, we present a mapping study focused on how machine learning has been applied to gaze data. We start by presenting our method in Section 2, then we present our results in Section 3, before we discuss the results in Section 4 and conclude in Section 5.

2 Method

In order to investigate the overarching question of 'how have machine learning techniques been applied to gaze data gathered from software development activities?', we carried out a systematic mapping study [15], by first breaking down our overarching question into the following three lower-level research questions:

- **RQ_1** What problems have been tackled by using machine learning on gaze data in software development?
- **RQ_2** How is machine learning used with gaze data in software development?
- **RQ_3** What are the open challenges in using machine learning on gaze data in software development?

[1] "Eye Tracking fully integrated and baked right into the very latest high performance gaming devices from Alienware, Acer and MSI". `https://gaming.tobii.com/products/laptops/` (Visited Feb 15, 2023)

```
1 SUBJAREA ( comp ) AND TITLE-ABS-KEY (
2 ( {debugging} OR {programming} OR {source code navigation}
OR {code browsing} OR {code search}
3      OR {code review} OR {code reading} OR {code
comprehension} OR {program comprehension} )
4 AND ( {eye tracking} OR {gaze} OR {eye movement} OR
{eye-tracker} )
5 AND ( {machine learning} OR {ML} OR {AI} OR {deep learning}
OR {artificial intelligence}
6      OR {classifier} OR {classification} OR {support vec
tor} OR {SVM} OR {decision tree}
7      OR {random forest} OR {RF} OR {naive bayes} OR
{naïve bayes} OR {NB} OR {linear regression} OR {LR}
8      OR {clustering} OR {imitation learning} OR {rein
forcement learning} ) )
9 AND ( EXCLUDE ( DOCTYPE , "cr" ) )
```

Figure 1: Search string used for data gathering. SUBJAREA limits the area to Computer Science. TITLE-ABS-KEY searches titles, abstracts, and keywords of publications. DOCTYPE with the option "cr" means "conference review". The highlighted parts show the keywords added in the second iteration of the search string.

We then composed the search string focused on software development activities, gathering of gaze data, and machine learning, shown in Fig. 1 (without the highlighted part which was added later). When we ran this search string on Scopus[2] on Jan 25, 2023, we got 50 papers in the search result. The first three authors then screened the title, abstract, and keywords of these papers using the following inclusion and exclusion criteria:

- **Include** paper if it employs eye-tracking, machine learning, and software development activity (SDA).
- **Exclude** paper if it is not in English, not available in full text, or is not a primary study. If a study described in a paper does not contain an experiment, then we did not consider it to be a primary study. However, if the study uses a pre-existing data set with gaze data gathered from an experiment, we considered this to be a primary study.

We employed a rule of minority inclusion when there was a disagreement on a paper. That is, even if only one author deemed that a paper should be included, we included it for further examination. Our rationale was that applying this rule would minimize the risk of missing a relevant study. After screening by all authors

[2]https://www.scopus.com

and the resolution of disagreements, 13 papers remained (i.e., 37 papers were excluded). Next, all three authors read the full texts of the remaining 13 papers, using the same inclusion and exclusion criteria from the screening. After the full-text review, we discussed any disagreement on inclusion and managed to reach a consensus for all discussed papers. In the end, 8 papers remained (i.e., 5 were excluded).

The first author then proceeded to collect the following data from the 8 papers: topic domain (using software development activities as codes), ML algorithms used, the purpose of ML use (a.k.a, prediction task), use of a pre-existing data set, input and output of the ML models, stimulus type, study setting, participant type, the number of participants, additional sensors used, limitations/threats to validity and future work.

Meanwhile, we informally reviewed past publications from the workshop venue - Eye Movement in Programming (EMIP). We found two papers that had the potential to be added to the final paper set but were not captured by the search string mentioned above. The reason was the use of the term "code reading" as a representative software development activity, which we had not included in our initial search string formulation. In addition, when we were coding the data gathered from the 8 papers mentioned above, we identified some frequently-used machine learning techniques, e.g. Random Forest. Hence, we refined the search string to include these techniques to potentially capture more papers of interest, the additions are highlighted in Fig. 1. When we used this search string on Scopus on Feb 1, 2023, we got 83 papers in the search result, with an overlap of 50 papers with the result of the previous search string, i.e., we found 33 new papers. We then went through the same process as described earlier with these 33 papers and ended up including 2 further papers, resulting in a final set of 10 papers.

Since there was strong inter-rater reliability (Kappa=0.84) between the first three authors during the first round of screening and we always reached a consensus after discussion throughout the screenings, we deemed it sufficient for the first author to proceed alone with the coding of the gathered data [3].

When coding experiment-related attributes such as the number of participants and participant type, we collected the information included in publications connected to the study described in the reviewed paper if it is not reported or omitted in the paper we reviewed. Otherwise, we used or prioritized the information reported in the reviewed paper. In some papers we reviewed, we found that their data sample was a subset of the data from an earlier study, e.g. to meet some custom research needs [8].

Table 1: **Overview of selected papers**. ICPC = International Conference on Program Comprehension, ISSRE = International Symposium on Software Reliability Engineering, ETRA = Eye Tracking Research and Applications Symposium, APSIPA ASC = Asia-Pacific Signal and Information Processing Association Annual Summit and Conference, JSS = Journal of Systems and Software, VISSOFT = Working Conference on Software Visualization, UMAP = Conference on User Modeling, Adaptation and Personalization, JCC = Journal of Cluster Computing, ICSE = International Conference on Software Engineering, EDA = Electrodermal Activity, HRV = Heart Rate Variability, EEG = Electroencephalogram, CSS = Closed Source Software, OSS = Open Source Software.

Paper	**Year**	**Venue**	**Use of data set**	**Stimulus type**	**Study setting**	**Participant type**	**No. participants**	**Extra sensors**
ASW22 [1]	2022	ICPC	No	Edu.	Lab	Mixed	16	EDA
APSM21 [3]	2021	ICPC	Yes	Edu.	Mixed	Mixed	20	-
HCC+21 [9]	2021	ISSRE	No	Edu.	Lab	Mixed	21	HRV
HN21 [8]	2021	ETRA	Yes	Edu.	Lab	Mixed	157	-
AO20 [2]	2020	APSIPA	No	Edu.	Lab	Students	66	-
VBMB20 [23]	2020	JSS	No	CSS	Workplace	Practitioners	37	EDA + Mouse
ZSPSY19 [27]	2019	VISSOFT	Yes	OSS	Mixed	Mixed	22	-
VR18 [22]	2018	UMAP	No	Edu.	Classroom	Students	84	-
LHJNL18 [11]	2018	JCC	No	Edu.	Lab	Mixed	38	EEG
FBMYZ14 [7]	2014	ICSE	No	Edu.	Lab	Practitioners	15	EEG + EDA + Camera

3 Results

We start by presenting the demographics of the papers we selected and then we present the results with regard to each research question.

3.1 Demographics

The demographics of the selected papers are listed in Table 1. Almost all papers (9 out of 10) were published in 2018 or later, with a peak in 2021 (3 papers). In terms of venue distribution, 8 papers were published in a conference or a symposium, and 2 in journals. The International Conference on Program Comprehension (ICPC) is the top venue with 2 papers. The majority of the papers (7) presented a study that included data collection, while 3 papers reused an existing data set.

[3]https://portal.research.lu.se/en/publications/applying-machine-learning-to-gaze-data-in-software-development-a-

The majority of papers (8) report on using educational material as stimuli, while the remaining papers (2) report on using code snippets from either open-source software (OSS) or closed-source software (CSS). When it comes to the choice of stimuli, the same type of material, e.g., educational material, is commonly used for both novices and experts, but there might be an increase in task difficulty or code complexity (or experts may get additional tasks). For studies with only practitioners, researchers tended to employ CSS material for the purpose of capturing realistic gaze behaviors in the workplace.

The majority of studies (6) were carried out in a laboratory setting, with a few (2) also including a workplace setting. One study was carried out in a classroom, and one study was carried out in a workplace. The studies in a mixed setting are usually those contrasting novices with experts in programming expertise, where novices are usually university students and experts are practitioners in the industry. Depending on the goals of the study, sometimes researchers asked both groups of participants to perform the tasks in the lab, while in some other cases, researchers conducted the experiments with experts in their offices.

The majority of papers (8) report using students as participants, which in a number of cases (6) were mixed with researchers and practitioners. The remaining 2 papers report using only practitioners. The number of participants ranges from 15 to 157, with a mean of 48, a median of 30, and a standard deviation of 45. Finally, half of the studies (5) used additional sensors; for Electrodermal Activity (EDA), Galvanic Skin Response (GSA, coded as a type of EDA), Electroencephalogram (EEG), Heart Rate Variability (HRV), mouse, and camera. The most common additional sensors were EDA (3) and EEG (2).

Table 2: **Heatmap of software development activity x ML technique**.

SVM and decision tree are frequently used for code comprehension

ML / Activity	**Code comprehension**	**Code review**	**Pair programming**	**Debugging**	**Total**
Support Vector Machine	4	0	0	0	4
Decision Tree	3	0	0	0	3
K-nearest Neighbours	1	1	0	0	2
Naive Bayes	1	0	1	0	2
Random Forest	1	1	0	0	2
Ensemble Learning	1	0	0	0	1
Graph Embedding	0	0	0	1	1
K-means Clustering	1	0	0	0	1
Linear Regression	0	0	1	0	1
Total	12	2	2	1	17

3.2 RQ1: What problems have been tackled by using machine learning on gaze data in software development?

To address this research question, we collected data on the software development activities studied. The top row of Table 2 gives an overview of the activities we found and the bottom row in the same table shows the total number of occurrences of an ML technique being applied to a data set gathered from this activity. Code comprehension is the most studied activity (12), followed by code review (2), pair programming (2), and visualization of debugging gaze data (1).

Table 3: **Heatmap of prediction problem x ML technique**.

SVM is frequently used for predicting programmer expertise and task difficulty

ML / Prediction	**Programmer expertise**	**Task difficulty**	**Mentally demanding code**	**Success of pair prog.**	**Quality of code review**	**Affect in code review**	**Code reading ability**	**Total**
Support Vector Machine	2	2	0	0	0	0	1	5
Decision Tree	1	1	1	0	0	0	0	3
K-nearest Neighbours	1	0	0	0	1	0	0	2
Naive Bayes	0	1	0	1	0	0	0	2
Random Forest	1	0	0	0	0	1	0	2
Ensemble Learning	0	0	1	0	0	0	0	1
K-means Clustering	1	0	0	0	0	0	0	1
Linear Regression	0	0	0	1	0	0	0	1
Total	6	4	2	2	1	1	1	17

Beyond activity, we also collected data on the purpose of using an ML technique. Across almost all papers (9 of 10), we found the purpose to be some kind of prediction task, summarized by the top row of Table 3 and counted in the bottom row of the same table. One paper, not shown in the table, used graph embeddings for mapping graphs into a 2D space. Note that multiple papers describe studies where several ML techniques were applied to the same data set [1, 2, 7, 22]. One paper may also describe more than one prediction task [11]. As such the totals in these tables (17) exceed the number of papers (10). The most common ML task was to predict programmer expertise (6), followed by task difficulty (4). After that

we saw prediction of mentally demanding code (2), success of pair programming (2), quality of code review (1), affect in code review (1), and code reading ability (1).

3.3 RQ2: How is machine learning used with gaze data in software development?

To address this question, we consider what ML techniques are applied on gaze data and how. From the heatmap presentations in Table 2 and Table 3, we see that Support Vector Machines (SVM) and Decision Trees are the two most commonly used techniques, followed by Random Forest (2), K-nearest neighbor (KNN) (2), and Naive Bayes (2). The preference for the top two techniques and Random Forests is due to their capability to handle large data sets [11,23], overfitting [3,11], and effective ranking/selection of features [3, 23], while the preferences for KNN and Naive Bayes is due to simplicity [7] and/or explainability [9].

To dig deeper into how the ML techniques are applied, we consider the ML techniques from a perspective of a model taking input and providing output, summarized in Table 4. The listed models correspond to the ML techniques listed earlier in Table 2 and Table 3. In the input column, we convey our interpretation of the details from the descriptions provided in the papers along with examples for clarification. We find a fairly high degree of heterogeneity, which may not be surprising given the available granularity/properties/dimensions of gaze data. However, in general, we see a tendency to use raw gaze data in combination with embedded feature selection algorithms (e.g., [1, 11]) to help pinpoint the prominent features for a specific task. In cases where several sensors are used (e.g., EDA, EEG) or data is gathered via other methods (e.g., answers to a questionnaire for participant performance assessment [2]), the data from the other sensors are incorporated in the training data set. Further, some researchers also used analysis results or derived metrics (e.g., CRQA attributes [22]) of gaze data as the input for training their models.

3.4 RQ3: What are the open challenges in using machine learning on gaze data in software development?

We address this research question from two aspects: limitations and challenges. We constructed the following themes from the limitations reported in the set of selected papers (not all papers reported limitations), sorted on occurrence in papers:

- **Limited sample size** (4 papers): the size of the sample is not large enough (perhaps despite being representative [3]) and thus may impact the applicability or generalizability of the model [1, 7, 23].

Table 4: **Overview of input, output and ML model used**.

Paper	Input	Model	Output
ASW22 [1]	line and fragment level gaze data, e.g., "features derived from fixation, saccade, pupil, scan-path"	Decision Tree, Ensemble Leaning	mentally demanding code fragments
APSM21 [3]	token level gaze data, e.g., "Single Fixation Duration, First Fixation Duration"	Radom Forest	programmer expertise (novice or expert)
HCC+21 [9]	biometric features and non-biometric features, e.g., HRV, scan time, code complexity, experience level	K-nearest Neighbour	code review quality of code regions
HN21 [8]	eye movement features, e.g., overall eye movement horizontally and vertically	Support Vector Regression	code reading ability
AO20 [2]	total fixation duration (TFD) and participants' answers to questions	K-means clustering, Decision Tree, K-nearest Neighbour, Support Vector Machine	expertise of novice programmer
VBMB20 [23]	gaze data + gaze shift data + EDA data + mouse logs, e.g., Euclidean distance and velocity derived from consecutive gaze samples	Random Forest	affect of the code review
ZSPSY19 [27]	eye movement trajectories and graphs of the software program, e.g., scanpaths	Graph Embedding (node2vec)	embedded space: positions in a 2D space and their distances to each other
VR18 [22]	Cross-Recurrence Quantification Analysis (CRQA) metrics, e.g., "average diagonal length (L) means the in-sync fixation paths"	Naive Bayes, Linear Regression	success of pair programming
LHJNL18 [11]	psycho-physiological sensors data (eye movement + EEG), e.g., total fixation duration	Support Vector Machine	programmer expertise and task difficulty
FBMYZ14 [7]	eye movement data + EEG + EDA, e.g., total fixation duration	Naive Bayes, Support Vector Machine, Decision Tree	task difficulty

- **Unbalanced data set** (3 papers): skewness in sampling, insufficient or absence of representation of certain cohorts in the data set [7, 23], e.g., lack of samples of professional programmers [3].
- **Binary categorization** (2 papers): specifically refers to the potential risk of over-simplification of programmer expertise and the task difficulty with a binary categorization method, e.g., novices vs. experts [9], and easy vs. hard [7].

- **Information trade-off** (1 paper): loss or compromise of certain information due to the use of an ML technique, e.g., the nuanced differences between two gaze trajectories became difficult to identify after having been mapped into a 2D space in the work by Zhang et al. [27].

Similarly, we created themes based on the challenges reported in the paper, as follows:

- **Experiment management**: the inherently difficult-to-control conditions of an eye-tracking experiment, potentially interwoven with other advanced sensors (e.g., EDA [1,7], EEG [7], or HRV [23]), and sensitivity to environmental variables such as illumination, the Hawthorne effect [12] (participants may attempt to act in a manner pleasing to the researcher), and learning effects while performing the tasks [7]. In addition, a mismatch between a laboratory setting and a natural work environment is most likely unavoidable.

- **Generalizability and validity of ML models**: ML models by nature are vulnerable to bias embedded in the training data set [2, 7, 23] and overfitting when trained on a small data set [1, 3, 7, 23]. This sometimes calls the validity and generalizability/applicability of the trained models in question. However, some researchers deliberately chose certain ML algorithms, e.g., Random Forest [3, 23], over others to mitigate this problem. Regardless, researchers still stated that due to the lack of diversity in their data and a small-sized sample, further validation [2, 11] or in-situ evaluation [9, 23] with a larger or unseen data set is needed or recommended.

Additionally, the challenge of getting access to professional programmers was also mentioned [1], as well as the challenge of finding the optimal parameters to derive metrics for training [22].

4 Discussion

Similar to the findings reported by Kuang et al. [10], we found that the majority of studies applying machine learning to gaze data are carried out with students, in a laboratory setting, and using educational material. As such, it could be questioned whether the reported results generalize to practitioners in the workplace, a concern also raised in the selected papers. A further concern is the small data sets, in the context of machine learning, used by the studies reported here. Small data sets limit the complexity of the deep learning models that can be developed [16, 20]. We find the community effort to publish more open data [13] and to curate larger data sets collaboratively [4] very promising for the future application of ML techniques. Still, despite these efforts, it may be challenging to compose really large data sets. We recommend that more attention is given to the use of

existing techniques for mitigating this problem, e.g., data augmentation [24, 26] or the generation of suitable training sets [25].

4.1 Opportunities for Future Research

Below we synthesize the future work stated in the studied papers as opportunities, supplemented with our own insights (sorted after occurrences in papers):

- **Extension of application** (4 papers): the possibility of applying the trained models to a similar sub-area [11, 23, 27] or a neighboring field rather than software development, e.g., computer science education [9].
- **Prediction task expansion** (4 papers): the aspiration to expand the prediction of the trained model to cover other similar or finer-grained tasks [2, 3, 7, 23], e.g., "predict where and when the eyes move across a line of code" [3] and detect "confusion and misinterpretation" [23].
- **Model validation and enhancement** (3 papers): this action concerns employing another data set to test the validity of the trained model [2], and two ways of enhancing the models with enriched data: training data diversification (e.g., [22]) and balancing [1]. One study may mention both enhancement methods. Both training data diversification and balancing increase the scale of the training data.
  - **Training data diversification** (3 papers): include new data from a completely new cohort or scenario [1, 2], or with new features of existing data points [22].
  - **Training data balancing** (1 paper): increase sampling of the current under-represented or hard-to-recruit participant cohorts [1].
- **System improvement** (3 papers): the envisioning of integrating the trained model into existing systems or tools to better support programmers [7, 11, 23].
- **In-situ evaluation** (3 papers): the intention to evaluate the trained model on data from a work environment [9, 23] or with a more realistic user study [27].

4.2 Threats to Validity

The typical limitations of a mapping study [6] relevant to the work presented here are the study selection process and the data extraction process. For the study selection process, the initial set of articles was based on a search string in an academic database. We chose a well-known database, which we believe is rather complete. The search string was iterated upon in order to find all relevant articles, but there may still be articles that we did not find with our search string. To mitigate this

concern, we have included extra search terms to find relevant articles from the EMIP community, as described in Section 2. The selection of studies was further performed by consensus of multiple authors. Given these actions and our own knowledge of the field, we believe that we have identified a reasonably good set of articles. For the data extraction process, the extracted data was iteratively discussed within the author group, e.g., based on previous research and knowledge of the research field. Since we decided to collect rather well-known, accepted and explicit types of data, such as the type of machine learning model, we believe that no large errors or misunderstandings have occurred in the data extraction and analysis.

5 Conclusions

This systematic mapping study aims at shedding light on the state of the art of applying ML techniques to gaze data in software engineering. We found a small set of papers matching our criteria but at the same time a growing trend in the number of papers being produced in this area. We identified the software development activities studied so far with eye-tracking and machine learning: code comprehension, code review, pair programming, and debugging. We further found that machine learning has been used for prediction of programmer expertise, task difficulty, mentally demanding code fragments, success of pair programming, quality of and affect in code review, and code reading ability. The most frequently used machine learning techniques are Support Vector Machines and Decision Trees. Overall for all the considered studies, the main limitations reported are limited sample size, unbalanced data set, binary categorization, and information trade-off, and we further found two overarching challenges in experiment management and generalizability/validity of machine learning models. These limitations and challenges set the stage for many opportunities for future work, e.g., including applications in neighboring fields like computer science education, or to expand into new prediction problems.

Acknowledgement

This work was partly funded by the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231), the Swedish Research Council (grant nbr. 2019-05658), and Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Amine Abbad-Andaloussi, Thierry Sorg, and Barbara Weber. Estimating developers' cognitive load at a fine-grained level using eye-tracking measures. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, pages 111–121, 2022.

[2] Zubair Ahsan and Unaizah Obaidellah. Predicting expertise among novice programmers with prior knowledge on programming tasks. In *2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 1008–1016. IEEE, 2020.

[3] Naser Al Madi, Drew Guarnera, Bonita Sharif, and Jonathan Maletic. Emip toolkit: A python library for customized post-processing of the eye movements in programming dataset. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–6, 2021.

[4] Roman Bednarik, Teresa Busjahn, Agostino Gibaldi, Alireza Ahadi, Maria Bielikova, Martha Crosby, Kai Essig, Fabian Fagerholm, Ahmad Jbara, Raymond Lister, Pavel Orlov, James Paterson, Bonita Sharif, Teemu Sirkiä, Jan Stelovsky, Jozef Tvarozek, Hana Vrzakova, and Ian van der Linde. Emip: The eye movements in programming dataset. *Science of Computer Programming*, 198:102520, 08 2020.

[5] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[6] David Budgen, Pearl Brereton, Sarah Drummond, and Nikki Williams. Reporting systematic reviews: Some lessons from a tertiary study. *Information and Software Technology*, 95:62–74, 2018.

[7] Thomas Fritz, Andrew Begel, Sebastian C Müller, Serap Yigit-Elliott, and Manuela Züger. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th international conference on software engineering*, pages 402–413, 2014.

[8] Hiroto Harada and Minoru Nakayama. Estimation of reading ability of program codes using features of eye movements. In *ACM Symposium on Eye Tracking Research and Applications*, pages 1–5, 2021.

[9] Haytham Hijazi, José Cruz, João Castelhano, Ricardo Couceiro, Miguel Castelo-Branco, Paulo de Carvalho, and Henrique Madeira. ireview: an intelligent code review evaluation tool using biofeedback. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 476–485. IEEE, 2021.

[10] Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. Towards gaxe-assisted developer tools. In *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2023.

[11] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. Mining biometric data to predict programmer expertise and task difficulty. *Cluster Computing*, 21:1097–1107, 2018.

[12] Jim McCambridge, John Witton, and Diana R Elbourne. Systematic review of the hawthorne effect: new concepts are needed to study research participation effects. *Journal of clinical epidemiology*, 67(3):267–277, 2014.

[13] Ian McChesney and Raymond Bond. Eye tracking analysis of code layout, crowding and dyslexia - an open data set. In *ACM Symposium on Eye Tracking Research and Applications*, ETRA '21 Short Papers, New York, NY, USA, 2021. Association for Computing Machinery.

[14] Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), 2018.

[15] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[16] Sarunas J Raudys, Anil K Jain, et al. Small sample size effects in statistical pattern recognition: Recommendations for practitioners. *IEEE Transactions on pattern analysis and machine intelligence*, 13(3):252–264, 1991.

[17] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25, 06 2020.

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

[19] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

[20] Andrius Vabalas, Emma Gowen, Ellen Poliakoff, and Alexander J Casson. Machine learning algorithm validation with a limited sample size. *PloS one*, 14(11):e0224365, 2019.

[21] Eva AM van Dis, Johan Bollen, Willem Zuidema, Robert van Rooij, and Claudi L Bockting. Chatgpt: five priorities for research. *Nature*, 614(7947):224–226, 2023.

[22] Maureen Villamor and Ma Mercedes Rodrigo. Predicting successful collaboration in a pair programming eye tracking experiment. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*, pages 263–268, 2018.

[23] Hana Vrzakova, Andrew Begel, Lauri Mehtätalo, and Roman Bednarik. Affect recognition in code review: An in-situ biometric study of reviewer's affect. *Journal of Systems and Software*, 159:110434, 2020.

[24] Sebastien C. Wong, Adam Gatt, Victor Stamatescu, and Mark D. McDonnell. Understanding data augmentation for classification: When to warp? In *International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–6, 2016.

[25] Raimondas Zemblys, Diederick C Niehorster, and Kenneth Holmqvist. gazenet: End-to-end eye-movement event detection with deep neural networks. *Behavior Research Methods*, 51(2):840–864, 2019.

[26] Raimondas Zemblys, Diederick C. Niehorster, Oleg Komogortsev, and Kenneth Holmqvist. Using machine learning to detect events in eye-tracking data. *Behavior Research Methods*, 50(1):160–181, Feb 2018.

[27] Li Zhang, Jianxin Sun, Cole Peterson, Bonita Sharif, and Hongfeng Yu. Exploring eye tracking data on source code via dual space analysis. In *2019 Working Conference on Software Visualization (VISSOFT)*, pages 67–77. IEEE, 2019.

PAPER III

DEVELOPERS' PERSPECTIVE ON TODAY'S AND TOMORROW'S PROGRAMMING TOOL ASSISTANCE: A SURVEY

Abstract

Software development is a complex activity that needs a lot of tool assistance. Over the years there has been a lot of effort put into development of automated assistance to help with activities such as detection of issues via program analysis, or refactoring of code. Recently, the landscape of developer tool assistance is being disrupted with the entry of AI tools, such as Copilot and ChatGPT, powered via Large Language Models. Other kinds of tool assistance, for instance, gaze-driven assistance, is around the corner. What are programmers' perceptions on tool assistance today? What do they see as good directions for the future?

In this paper, we present the results of a survey where we asked developers about their programming practices, experience with program analysis, and attitudes and views on enabling technologies, like AI and eye-tracking. We received 68 replies from a diverse group of developers from 12 different countries. We found that 50% of the participants use program analysis and that many participants (28) already use AI-enabled tools for programming. We found that our participants were positive toward AI-powered tools, neutral toward eye-tracking, and negative toward gamification. We discuss these and other findings and point out directions for future work.

1 Introduction

Software development is a complex activity and there are numerous efforts to provide tool assistance to programmers, for instance, automatic discovery of defects and vulnerabilities via program analysis [22]. We know from past work on programming tool assistance that while a technology may provide benefits it may also bring drawbacks. For instance, program analysis has been recognized as providing promising tool assistance, with several reports of use in practice [5, 17, 18, 25], but at the same time there are numerous reports of usability issues [5, 7, 9–11, 18].

Recently the landscape of tool assistance for developers has been disrupted by the introduction of AI-based tool assistance, for instance, Copilot [15] and ChatGPT [14]. We do not have to imagine the generation of code from natural language, we already have it – even though much may remain to utilize this technology in an efficient way (e.g., security issues with LLMs [2]). Other technologies that may be disruptive will probably arise in the future. For instance, eye trackers may slowly be making their way into the development environment [8, 19, 23]. With the addition of devices, like the Apple Vision Pro [1], it is not too distant to imagine the possibility of a programming environment with an integrated eye-tracker.

To effectively explore designs in the area of programming tool assistance, we need input from programmers, especially when the landscape is being disputed by entry of new technology. There are examples of surveys of developers focusing on specific tools in the programmer workflow, for instance, Liang et al. [12] focusing on AI-based programming and Do et al. [7] focusing on program analysis, and surveys focusing on specific groups of programmers, e.g., Nguyen-Hoan et al. [13] focusing on scientific programmers. Among these examples of surveys, only Liang et al. [12] occurred after the disruption of AI programming. We see a need for a broader focus where AI programming is considered together with other tool assistance. We also see an opportunity to ask developers about how they relate to tool assistance available today, as well as, possible future directions for tool assistance.

In this paper, a survey is presented where we reach out to developers to collect their view on programming tool assistance, both with regard to today's tools but also with regard to what may be developed in the future. We focus on the following research questions: (**RQ$_1$**) *What are developers' perception on tool assistance today?* and (**RQ$_2$**) *What are developers' perception on directions for future tool assistance?* To address these questions, we created a survey where we asked developers about their programming practices, experience with tool assistance like program analysis, and attitudes and views on enabling technologies or strategies like AI, eye-tracking, and gamification. We received 68 replies from a diverse group of developers from 12 different countries. We found that 50% of the participants use program analysis and that many participants (28) already use AI-enabled tools for programming. We further found a positive sentiment toward AI-based assis-

tance, neutral toward eye-tracking, and negative toward gamification. We discuss our findings and point out directions for future work.

2 Method

To address $\mathbf{RQ_1}$ and $\mathbf{RQ_2}$, we designed a survey to better understand professional developers' programming practices and experience with program analyzers, and their perception on a selection of enabling technologies; AI, eye-tracking, and gamification[1].

2.1 Data Collection

The survey contains 26 questions that were drafted by the first author and reviewed by the rest of the authors. A master question together with its sub-question set, which was prompted to participants depending on their answers to the master question, are collectively counted as one question in our case. For example, "Can you type out one or more program analysis tools that you are aware of?" and "Have you ever used any program analysis tools?" are both sub-questions to "Have you heard of program analysis?". In particular, we incorporated some questions from a recently published paper by Peitek et al. [16] to capture the expertise of the participants more accurately, compared to using years of programming experience as an indicator. The survey was created and executed using an internal survey tool named SUNET offered by the university.

Since the understandability of analysis results stands out as a barrier to adoption in the literature [3,4,6], we designed questions specifically asking participants whether or not their English proficiency has an impact on their understanding of error messages or program analysis results. After that, the survey was pilot-tested with two research colleagues; one senior researcher and one PhD student with several years of industrial experience as a professional developer. The survey took them 13 and 15 minutes to fill out, respectively.

The survey was then advertised by the first author via LinkedIn and shared by the rest of the authors who own an active account there. Some of the colleagues and friends of the first author also shared the LinkedIn post to help spread the word. The ad was further shared by the first author to a Swedish network of academics. In addition, the first and second authors reached out to professional programmers within their circles via email and social media such as WeChat and Facebook. When it comes to individual academics, the authors specifically approached those who were known by them to have worked or still be working part-time in the industry.

[1] The replication package is available at: doi:10.5281/zenodo.10777303. It includes two versions of survey questions, a Python script used for analyzing the quantitative data, and a coding book used for analyzing the qualitative data

Based on the first few responses, the first author revised the survey into a new version with a separate link. The new vision had five questions added, four to cover AI-powered developer tools brought up by the first few respondents, and one to serve as a baseline for programming experiences. Some of the wording of questions was also adjusted based on the answers of those respondents, e.g., we added alternative names 'code analysis' and 'software analysis' for the term 'program analysis', in the question asking participants whether or not they have heard of this concept.

This created two versions of the survey. The latter had 5 more questions added but the rest untouched. Making this kind of necessary modification to fit into reality was also shared by some other researchers (e.g., [24]) in our field. The first author updated the survey link wherever it was possible. In the end, 68 participants responded to the survey. The first version received 29 responses and the second version 39 responses. When reporting the result from the survey, we refer to participants in the first survey using the prefix 'pa', e.g., pa1, and participants from the second survey using the prefix 'pb', e.g., pb2.

2.2 Data Analysis

For analyzing the survey, the first author produced a Python script to process the quantitative part (e.g., the questions with answers on a 5-point Likert scale) of the data; for the qualitative part (e.g., open-ended questions with free text answers), we applied bottom-up thematic analysis to the answers of the open-ended questions related to program analysis and AI/eye-tracking/gamification. Some participants answered the questions in their native languages, e.g., Chinese and Swedish. The first author translated such texts into English before analyzing them. The first author coded the qualitative data using open code [20], and then the second and third authors reviewed the coding. The rest of the texts, since most of them are succinct and straightforward, the first author summarized and weaved them into the results where most appropriate.

3 Results

In this section, we report the results on participants' demographics, programming expertise, programming practices, and perceptions of enabling technologies.

3.1 Programmer Demographics

Residence

The survey reached participants residing in 12 countries. The top 4 countries are Sweden (33), China (18), Australia (6), and Germany (3), with the rest 8 countries each with one respondent. The predominant native languages that participants

speak are Chinese (29) and Swedish (25), with a long tail of 11 other languages spanning from English (3) and German (2) to Sinhala (1) and Bahasa Indonesian (1). The participants include 57 males, 7 females, and 1 non-binary, with three preferring not to disclose. Their average age is 35.

Education

All participants have an education in STEM (Science, Technology, Engineering, and Mathematics): 63% of them had it solely in Computer Science/Software Engineering/Data Science/Machine Learning, 31% in the broad areas of STEM, and 6% with a mixed academic background, namely, they also have other degrees in non-STEM fields. None is with a completely non-STEM background.

Occupation

In terms of profession, the majority (66%) of participants are professional developers in the industry: 41% software engineers or developers, 15% tech leads or managers, 6% DevOps, 3% QA/testers, and 1% data scientists; the rest are mainly from academia: 21% PhD students, 3% professors, 3% researchers (e.g., postdoc), 1% teaching staff and 1% graduates. Interestingly, among the category of 'Other' (4%), participants specified that they are data engineers, both academic and practitioner, or senior management/leadership (e.g., CEO). Regarding occupation, 43% of the participants work in multinational corporations, 28% at small or medium-sized companies, 28% at universities or other educational institutes, and 1% is still a student or learner.

Programmer Identity

We asked participants whether they identify as programmers, 85% of the participants responded that they do, while 15% do not. The reasons for not identifying vary. Some participants link the practice of coding to the eligibility of being called a programmer, either context-wise or frequency-wise. When they are just *"scientific programming"* (pa8), or *"casual coding"* (pb36), they express that they do not qualify as a programmer, or when they do not code that much anymore, even though they used to be a professional software developer. Some participants link it with the nature of their job, e.g., *"devops scripting"* (pb6), *"engineer"* (pb15), and *"hacking"* (pa22), or with the relatively low level of their programming skill or being self-taught. In general, these non-programmer-identifying participants tend to equate programmers to professional software developers and deem that the latter has to be very capable of programming.

Programming Skill

In terms of programming skills, 21% of the participants reported that they are at expert level, 54% advanced, 21% intermediate, and 4% beginner. When asked about their confidence in programming, 49% reported very confident, 43% somewhat confident, 7% unsure or depends, and 1% not confident at all. Compared with peers, 15% report themselves as much better at programming, 40% better, 35% same, 9% worse, and 1% much worse. Compared with an expert with ten years of programming experience, 6% report themselves as much better, 15% better, 32% same, 34% worse, and 13% much worse. Among the 39 participants (57% of the entire participant population) of the second version of the survey, their average years of experience in programming is 11 years.

According to Peitek et al. [16], self-reporting proficiency by programmers is a better indicator of a programmer's coding skill measured in programmer efficacy (correctness/time spent) than the years of experience in programming. Therefore, it is reasonable to say the majority of the participants in our study are advanced programmers.

Programming Languages and Paradigms

The top three primary programming languages used by the participants are Python (31%), Java (29%), and the C family (27% in total, consisting of C 12%, C++ 9% and C# 6%), followed by Javascript (7%), Go (6%), and other 11 programming languages such as Swift, Ruby and so on. Besides their primary programming languages, the participants are also familiar with some other languages. The top-mentioned other languages remained to be the most popular ones: the C family, Python, and Java, except Rust which did not surface in the primary language cohort.

For 68% of the participants, their primary language is their favorite language. When asked about what they liked or did not like about the primary language, many participants highlighted the factor of the resources available or the ecosystem (libraries/tool support/developer community) for a language, along with other factors including easy-to-learn syntax, level of abstraction, restrict type system, match with the goal of the task, suitability for the scale of the project, and wide-range adoption.

Regarding programming paradigms, 92% of the participants are familiar with objected-oriented programming, 59% with imperative programming, 44% with functional programming, and 16% with logic programming.

Development Activities

On average, our participants worked for 43 hours per week in the past three months. Unsurprisingly, as shown in Figure 1, they spent most of their time on programming (28%), followed by meetings (20%), other (13%), learning/training (12%),

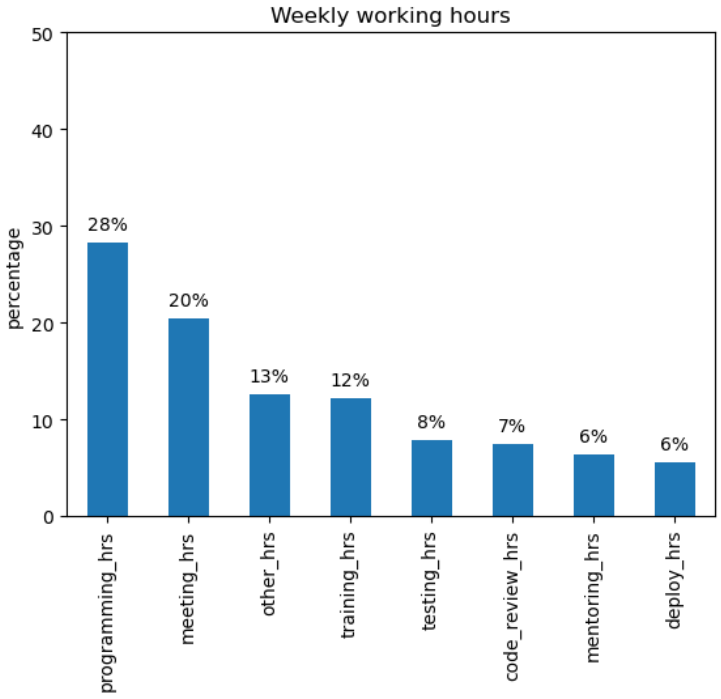


Figure 1: Distribution of weekly working hours.

testing (8%), code review/process (7%), mentoring (6%), deployment/operations (6%). In the 'other' category, we asked participants to specify the activities that were related to programming or software development. Participants mentioned communication with co-workers (both in a task context, e.g., clarifying requirements, and social context, e.g., Fika [21]), documentation, and debugging. Participants from academia brought up research activities such as reading and writing. Others stated concrete activities closely tied to the nature of their job, for example, spending time with customers and modeling threats.

When asked about the frequency of actually writing code per week during the past three months, 47% of the participants answered almost every day each week, 19% of them answered 3–4 days per week, 18% 1–2 days per week, 6% read code mostly, 3% read code sometimes, 6% read code occasionally, and 1% never need to read or write code.

Development Environment

Regarding operating systems, 56% of the participants programmed in Windows, 41% in Mac OS, 67% in Linux, and 3% in other environments. In terms of the

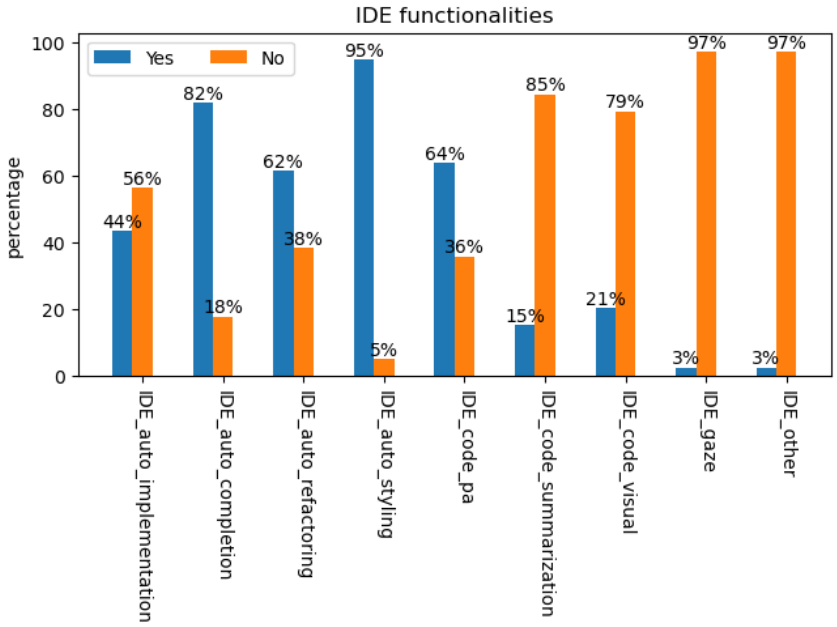


Figure 2: Functionalities of the IDEs/editors.

IDEs/editors that the participants are using, the top five mentioned are Visual Studio Code (56%), Vim (19%), Visual Studio variants (15%), IntelliJ (15%), and Pycharm (9%), followed by many other IDEs/editors such as Eclipse (7%), Jupiter (4%), Emacs (4%), Squeak/Smalltalk (3%), Xcode (3%), Spyder (3%), JetBrains (3%), just to name a few. Regarding the features provided by the IDEs/editors (Figure 2), 44% of the participants ticked auto-implementation, 82% of them ticked auto-completion, 62% ticked auto-refactoring, 95% ticked auto-styling, 64% ticked program analysis or code analysis, 15% ticked code summarization, 21% ticked code visualization, 3% ticked gaze control and/or gaze visualization, and 3% ticked other features. One participant who selected the other category stated it was referring to *"version control management"* (pb15). When it comes to version control or version management, 96% of the participants used Git, 3% of them used SVN, 1% used Mercurial, and 4% used other tools.

3.2 Perceptions on Today's Tools (RQ_1)

We report participants' experience with program analysis, AI-enabled tools, their main pain point with programming, and the language factor in understanding error messages.

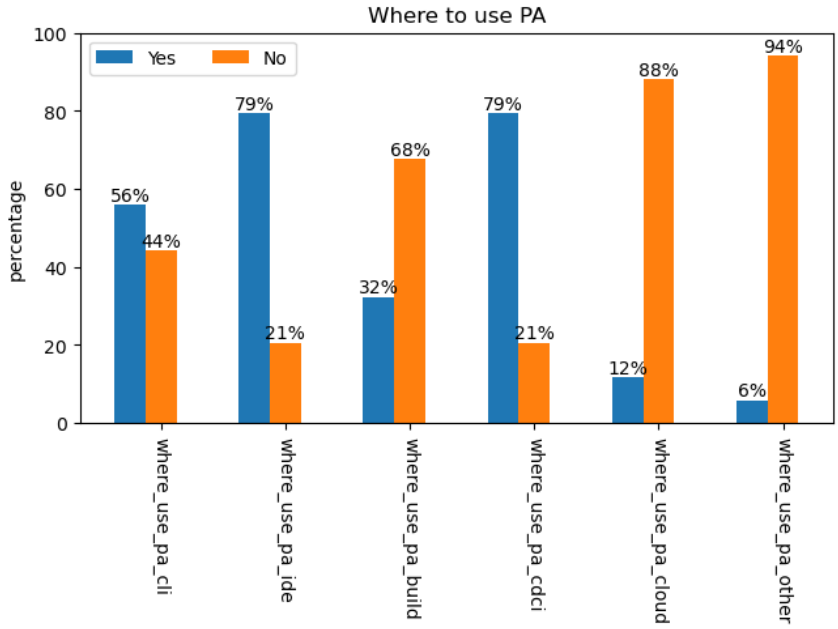


Figure 3: Where to use program analysis.

Experience with Program Analysis

More than half of our participants (59%) have been exposed to program analysis. Among the participants who know what program analysis is, 85% of them (34) have used program analysis tools. Among those who used such tools, 74% are still using them in their current work. The top-mentioned program analysis tools are Coverity 19%, unspecified/general Linters 12%, SonarQube 10%, Pylint 10%, Findbugs or Spotbugs 9%, Snyk 6%, Flake8 4%, Cppcheck 4%, PMD 4%, and Semgrep 3%.

Among the 34 participants who reported having used program analysis tools, 53% of them used these tools before compiling or building their programs, 53% of them used during, 71% after, and 24% selected other. In relation to version control, 74% of participants used program analysis before pushing their code to the repository, 50% during, 56% after, and 12% selected other. Regarding where to use program analysis (Figure 3), 56% of participants used these tools in the command line interface, 79% in IDEs, 32% during building, 79% during Continuous Integration and Continuous Deployment (CI/CD), 12% in the cloud, and 6% selected other.

The reasons why participants do not use program analysis tools include reasons like there not being a need, redundant to the built-in analysis functions in the IDEs,

costly to set up (time and/or money wise as some tools are premium or close-sourced), low familiarity, noisy results, and organizational policy.

When asked about their first-time user experience with program analysis tools, some were impressed by what it can do with a positive sentiment, whereas some had a mixed or even a negative sentiment due to the following factors: hard to set up or make it work, verbosity of analysis output, lack of configurability, incapability of detecting logic errors and decreasing usefulness for an advancing programmer. For example, participant pa22 responded, *"I received a cppchecker dump from a Jenkins job and felt despair"*. While some did not recall how they got to experience program analysis in the first place, others depicted that it was introduced to them in either academic scenarios (e.g., when submitting assignments for a course), industrial ones (e.g., their first job as a software developer), or through the embedded functions of developer tools (e.g., IDEs, Gerrit, Jenkins) that they were using (e.g., suggested refactoring). The tools they used for the first time range from syntactic linters to memory leak checkers (e.g., Valgrind) and security vulnerability detectors (SonarQube).

Among the 40 participants who had exposure to program analysis, 28% strongly agree with the usefulness of program analysis, 50% agree, and 22% are neutral. None strongly disagree or disagree.

Finding 1: Half of participants (50%) use program analysis, and about 60% have been exposed to it. Those who are familiar with program analysis all find it useful. The main frustrations reported concern setup costs, quality of results, and presentation of results.

Experience with AI-enabled Tools

The AI-related questions were only applied to the 39 participants of the second version of the survey. The majority of these participants (77%) have used AI tools for programming, while 23% had never tried. Among the 30 participants who had experience with AI tools (Figure 4), 72% used ChatGPT, 23% used Copilot, and 3% used other similar tools. For those who did not use such tools, the main reason was that they had not encountered the need yet - either they were fluent with the current programming tasks, or they did not see a programming task where such tools would help. Some also shied away because of the time and money costs of setting up and learning the tool, or it was simply *"not available on company's workstation"* (pb36). Meanwhile, those who write code for publications were concerned about potential copyright risks.

Among those 30 participants who have experience with using AI-enabled developer tools, 27% strongly agree with the usefulness of such tools, 53% agree, 10% neutral, 3% disagree, 7% strongly disagree. The concrete tasks participants used these tools for are quite heterogeneous, ranging from generating comments,

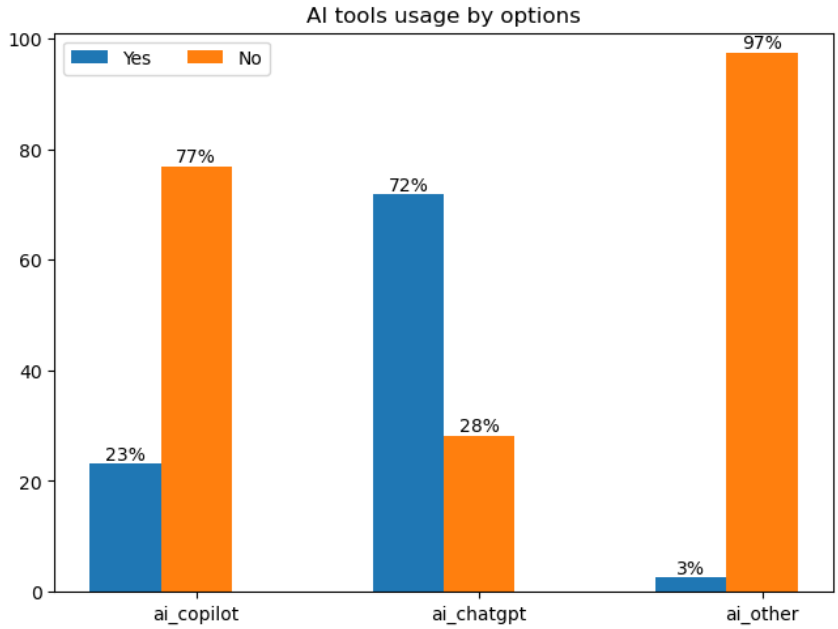


Figure 4: Usage of AI tools.

test data, utility code, scripts, and skeletons, to querying about APIs, libraries, debugging, prototyping, and implementing a specific program. Academic programmers also used it with research motives, examining the possibility of integrating these tools into teaching and instructing students on how to use them properly.

When asked about their views on the pros and cons of these tools, many participants highlighted a boost in productivity, providing ideas, being interactive, and reducing programmers' fear of switching programming languages. The cons focused on the code generated not being good enough (thus needing time to verify or modify) and fostering dependence and hindering the learning of programmers; some were also concerned with data leakage and energy consumption of training and running the machine learning models behind the scenes.

Finding 2: Among the participants asked about AI-enabled tools (39/68), a majority (77%) have used AI tools for programming, and among these participants a majority (80%) is positive about the usefulness of these tools. Participants report using AI tools for a broad range of programming tasks and find these tools to help with productivity, creativity, and exploration of new areas. Participants not using these tools report a lack of need, lack of availability, or cost.

Main Pain Points with Programming

The main pain point in programming varies for each participant. It could be at many points of a programmer's workflow or associated with many software development activities. For the former, the answers range from requirements and designing to collaborating with other stakeholders (e.g., other programmers or product managers), tooling, coordinating dependencies, and managing backlog and priorities. For the latter, the answers cover reading a code base, understanding APIs, libraries, and frameworks, coding from scratch, setting up the environment, long-time building or compilation, version control, testing, debugging, and refactoring. Others mentioned the shortage of time for programming (thus losing fluency in a programming language), the fatigue from persistent programming, and the demand to keep up with the technologies needed, while some stated that they just enjoy programming and have nothing to complain about.

Finding 3: Our participants report a broad range of pain points, from frustration with colleagues to frustrations with tools (e.g., cumbersome setup, long build times, version control). The most prominent reported pain point is code comprehension (e.g., APIs, libraries, and frameworks).

Is Native Language a Factor?

When asked about whether or not their English proficiency has an impact on their understanding of general error messages generated by programming or developer tools, 9% of the participants indicated high impact, 26% some impact, 12% low impact, 12% little impact, and 41% no impact. For program analysis results, our participants indicated that it may be a factor, with 3% of these participants indicating a high impact, 38% some impact, 3% low impact, 24% little impact, and 32% no impact.

While admitting there are some impacts from the English proficiency, some participants detailed that it is usually because the phrasing of analysis results *"are not following Natural language rules"* (pb50) when they encountered confusing outputs. But if it does happen, they can still search on the web for a resolution,

the same as what they will do with other tools. Sometimes, the outputs of a program analysis tool can be non-textual such as a diagram, heatmap, or recoloring of something in the editor, then it may be more challenging to search the web.

Also, our participants tend to express that if programmers are fluent in programming, their English proficiency is sufficient; otherwise, they are not able to do other programming tasks since English is the *"de-facto language"* (pa44) for programming and communication in this community. Nonetheless, some participants stated that terminology in general should be avoided in analysis outputs, especially in the case of documentation translation, as many terms are not even stabilized in the other language.

We specifically looked into the data of six participants (pa4, pa23, pb17, pb20, pb 23 & pb29) whose native language is not English and with relatively lower proficiency (one elementary proficiency and five limited working proficiency) in English. Surprisingly, only one of them chose high impact, the rest were inclined to low to no impact (one low impact, two little impact, and two no impact). With the four of them who had exposure to program analysis tools, this trend somehow still resided - one chose some impact, two little impact, and one no impact. We further checked the levels of their programming skills. It turns out their programming skills vary: one beginner, one expert, one advanced, and three intermediate. It was the expert who chose high impact for general error messages, but little impact for program analysis results and one intermediate who chose little impact for the former and some impact for the latter.

Finding 4: Our participants' responses suggest that English as a non-native language may have an impact for understanding of general error messages (35%) and program analysis results (41%), but they also report a view on English as closely intertwined with programming as the defacto language.

Summary

We summarize our findings for $\mathbf{RQ_1}$ as follows:

What are developers' perception on tool assistance today? ($\mathbf{RQ_1}$)

Program analysis is broadly used in practice and the majority of users find it useful while reporting frustrations with setup and results. AI-based tools are making a big entry, the majority of participants asked about this topic report use for a range of programming tasks. Our participants report a variation of pain points, with code comprehension being the most prominent. English proficiency is indicated as impacting understanding but English is at the same time reported as part of understanding in programming.

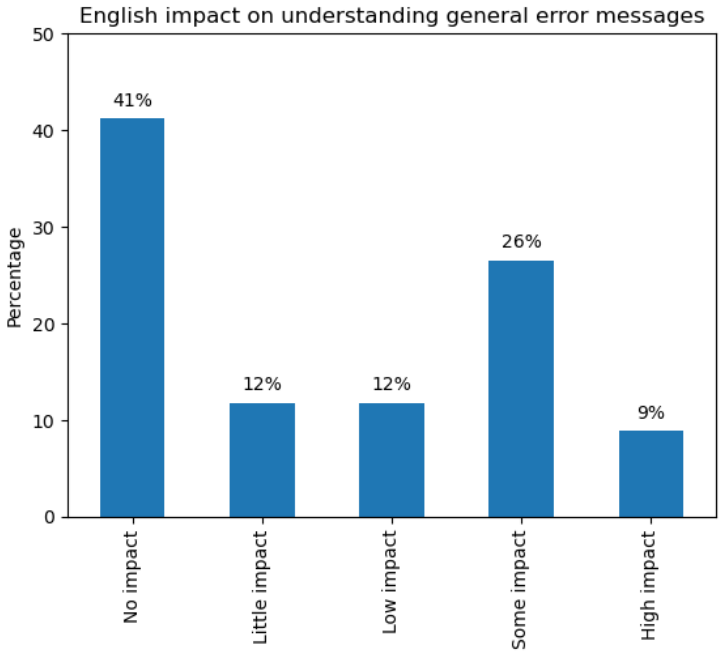


Figure 5: The impact of English proficiency on understanding general error messages.

3.3 Perceptions on Tomorrow's Tools (RQ_2)

We surveyed participants' attitudes toward three enabling techniques or strategies for future programming tools; AI/ML, eye-tracking, and gamification.

Attitude Toward AI or Machine Learning (ML)

Our participants are positive about AI/ML-assisted tools, with 18% indicating that they are very positive, 52% positive, 20% neutral, 10% negative, and none very negative. The majority of the participants believed in or were open to embracing AI/ML, but some participants commented with doubts and concerns. Several participants stated that they do not trust AI/ML enough and believe it is a bit of a hype (pa37, pb12).

We asked whether participants thought AI/ML could improve the quality of program analysis. Our participants mentioned multiple aspects: suppressing false positives (pa22), filtering out noises or less significant outputs, and facilitating understanding of both analysis outputs and analysis methods. Some expressed that they did not see any space in program analysis for improvement with AI/ML

(pa18), or there is yet a large gap to leap over to link AI/ML and program analysis (pb12), pending on how it will be implemented (pb55).

Other participants pointed out that the success of AI/ML in helping program analysis depends on whether or not the flaws with current AI/ML applications, such as verbosity and inaccuracy (confidently reporting incorrect answers), will be addressed (pa52, pb45 & pb50). Several participants were content with (pa52, pb16), or at least preferred to still have access (pa43), to the short and succinct format of raw outputs produced by the program analysis tools. This implies that there might be a fine line to draw when it comes to designing, or manipulating, the presentation of analysis results.

> **Finding 5**: The majority of our participants are positive to AI-enabled tools (70%), but also mention it as a hype. When asked about using AI/ML to improve program analysis, our participants saw several possibilities (e.g., suppressing false positives) but also expressed a preference to see the raw output due to uncertainty with inaccuracy with AI/ML.

Attitude Toward Eye Tracking

Our participants are neutral, with a noticeable inclination toward a negative sentiment, about eye tracking; 2% very positive, 12% positive, 50% neutral, 28% negative and 8% very negative. Some participants perceived eye-tracking as invasive, if it is going to monitor or record their gaze, and thus had concerns about their privacy being violated or revealed to employers (pa22, pa43 & pb12). Also, some participants deemed eye trackers as *"chunky"* (pa37) or *"bulky"* (pb52) hardware, which implies a burden. Some suggested the solution should focus on the code rather than the humans in terms of providing support to developers (pa22 & pb15).

> **Finding 6**: The majority of our participants are neutral toward eye-tracking (50%), with a lean toward negative (36%). They report concerns about privacy with the collection of gaze data and concerns about the equipment being burdensome.

Attitude Toward Gamification

Participants' sentiments toward this technique are more divided or polarized; 2% very positive about gamification, 22% positive, 30% neutral, 28% negative, and 18% very negative. Participants appear to either like it or hate it. While some participants complain this technique is *"childish"* (pa18, pa37), *"distracting"* (pb12, pb52), or "patronizing" (pb10), some participants believe it is *"promising"* (pb45), *"a good idea to encourage developers"* (pa42) and *"very interactive"* (pb55). Others think it may work for others although not for them, depending on a team's

culture or individual preference, or it can be useful, conditioned on how it is implemented, e.g., with good balance.

> **Finding 7**: Our participants' view on gamification is primarily negative (46%), but also polarized; they either like it (e.g., promising) or hate it (e.g., patronizing).

What Can Help in the Future?

We asked participants what technology they thought would improve their programming experience. Many participants pointed to AI, e.g., *"Large Language Models"* (LLMs) and *"Generative AI"*. Some participants were more specific, replying with ChatGPT and Copilot alike agents. They listed features such as auto-completion, auto-refactoring, providing analysis, feedback or summary, and suggesting fixes or better alternatives, which they assume would be either powered or enhanced by AI.

While the promise of AI technology was acknowledged, participants also indicated that the current AI-enabled tools cannot handle more sophisticated programming tasks, and thus improvement is needed. Some participants stressed the other party of this task - programmers. They deemed that daily practices or more bad code written by programmers would help them master the skill, e.g., *"Its the amount of bad code, compilation errors, bad architecture decisions, technical debts help you improve experience in programming"* (pa14).

> **Finding 8**: Our participants point toward AI-based tool assistance as a direction for the future, while also indicating that this technology needs to improve to assist with more sophisticated tasks. They also touch upon the risk of AI-based assistance taking away opportunities for learning.

Summary

We summarize our findings for $\mathbf{RQ_2}$ as follows:

What are developers' perception on directions for future tool assistance? ($\mathbf{RQ_2}$)

Our participants are generally positive toward AI-based assistance, and see opportunities for several improvements of existing tools using this technology. They are wary of eye-tracking, with concerns about privacy and inconvenient equipment, while being primarily negative about gamification. They see AI-based tools are the main direction to focus on for the future.

4 Discussion

We see a broad use of both program analysis and AI-based tool assistance. The usability problems of program analysis are similar to those reported in the literature, e.g., issues connected to the quality of the results and their presentation [10, 11]. AI-based tools, like ChatGPT and Copilot, are used for a broad range of programming tasks, and the main concern mentioned for AI-based tools concerns their reliability. Developers experience several points of frustration in their work, ranging from issues connected to the work environment to issues with tools. The most mentioned pain point in the survey was code comprehension.

For future directions, developers primarily see AI-based tools as the future, while they are wary about eye-tracking and negative about gamification. We note that our participants likely are the most familiar with AI-based tools, given the broad adoption and buzz around tools like ChatGPT, while few have seen an eye tracker. For gamification, we speculate that there are plenty of examples available in mainstream applications of gamification and participants may more easily imagine scenarios they relate to when responding to our questions in the survey.

4.1 Directions for Future Work

We see a couple of directions for future work:

Improved support for code comprehension: This was the most mentioned point of frustration in the survey result regarding frustrations with today's programming tools. We see this as an indication that there is more room for improvement in this area.

Improved reliability and integration of AI-based tools: Reliability of AI-based tools was the main concern mentioned in connection to AI-based tools. We see an opportunity to explore how to best integrate AI-based tools into the developer workflow.

Continued exploration of eye tracking with a focus on what developers want: The concept of eye tracking is largely novel and foreign to developers, and the notion of collecting gaze makes developers wary. We still think this direction is worthy of exploring but we encourage to proceed in this direction with care for what developers want.

4.2 Reflections on the Method

We encourage other researchers to plan for pilot tests of a survey with more than one participant from the target population, if possible before publishing it. Responses from participants with distinct backgrounds and experiences may help researchers rethink the design of their survey in terms of phrasing and logic and make necessary adjustments. It may also help if the researchers formulate a strategy for advertising, e.g., leave the advertisement with organic traffic for the first

two days and use it as an opportunity to check if there are any last-minute revisions needed before actively driving traffic to the survey such as sending group emails. Further, investigating the channels that you intend to use can be beneficial. For example, we did not know the link in a post on LinkedIn could not be edited after it was published. In terms of securing responses, in our case, we think approaching potential participants individually through one's network is the most effective way to achieve that.

4.3 Threats to Validity

We identify the following threats in our study:

Construct validity: Our sampling is skewed toward developers in Sweden and China. This is mainly because the authors adopted convenience sampling to gain traffic. This presents a threat to the diversity of the samples. However, the residence country is not a primary attribute that we look into for our participants. We also mitigated this threat by reaching out to prospective participants via a wide range of channels, including LinkedIn, Slack, Facebook, WeChat, email, and in-person invitations. It is further compensated by the fact that developers from these two countries were not well-represented in our study domain.

Content validity: The survey questions were discussed among the authors and reviewed by some colleagues. The authors are subject matter experts in the study domain and with the method. Most colleagues are representative of the target population. One colleague is not a programmer himself so it provides face validity to our survey. Their feedback was incorporated to revise the questions.

Criterion validity: The questions were organized in logical order with different sets of questions to capture the characteristics of different groups of the population. We also devised branching logic to orient participants to different sets of questions if they do not fulfill the criteria.

External validity: For the survey, the skewed sampling toward developers who reside in Sweden and China poses a threat to generating the findings to a broader population who do not share these characteristics. However, some findings from the qualitative data align with what we see in the literature, e.g., some reasons for not using program analysis tools and concerns with eye-tracking and AI/ML. Further, the sampling is deliberately to represent practitioners (including practitioners in academia or shifted to academia), so the findings do not apply to another distinct population, namely, novice programmers or students/learners.

5 Conclusions

We surveyed 68 developers to investigate their perception of the current and future programming tool assistance. Our inquiry covers their programming practices, experience with program analysis, and attitudes and views on enabling technologies

- AI/ML, eye-tracking and gamification. We found that 50% of the participants use program analysis and that many participants (28) already use AI-enabled tools for programming. The participants are positive with AI/ML, neutral with eye-tracking and negative with gamification. We suggest further work to be done to improve support for understanding code, enhance reliability and integration of AI-powered tools, and explore how eye-tracking better fits into developers' wants.

Acknowledgement

We express our gratitude to all the participants who took part in the survey or the pilot tests. We thank Diederick C. Niehorster for providing feedback on the survey questions and the paper. We thank Anton Risberg Alaküla for providing feedback on the survey questions and pointing us to related data. We thank Christofer Rydenfält, Lo Heander and Andreas Bexell for the fruitful discussion when we were selecting the study method and for helping us spread the word about the survey. We thank Shanton Chang and Nickole Li for helping advertise our survey through their platforms. We thank Ola Stjärnhagen for the survey tool support.

The authors would further like to thank the following funders who partly funded this work: the Swedish strategic research environment ELLIIT, the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231), the Swedish Research Council (grant nbr. 2019-05658), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

[1] Apple. The Home View on Apple Vision Pro, 10 2023. Retrieved Feb 8, 2024 from `https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/`.

[2] Owura Asare, Meiyappan Nagappan, and N Asokan. Is github's copilot as bad as humans at introducing vulnerabilities in code? *Empirical Software Engineering*, 28(6):129, 2023.

[3] Titus Barik, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. Do Developers Read Compiler Error Messages? IEEE, 5 2017.

[4] Brett A. Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J. Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, Janice L. Pearce, and James Prather. Compiler Error Messages Considered Unhelpful. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education*, 2019.

[5] Maria Christakis and Christian Bird. What developers want and need from program analysis: an empirical study. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE '16, page 332–343, New York, NY, USA, 2016. Association for Computing Machinery.

[6] Paul Denny, James Prather, and Brett A. Becker. Error Message Readability and Novice Debugging Performance. pages 480–486, 6 2020.

[7] Lisa Nguyen Quang Do, James R. Wright, and Karim Ali. Why do software developers use static analysis tools? a user-centered study of developer needs and motivations. *IEEE Transactions on Software Engineering*, 48(3):835–847, 2022.

[8] EIZO. Eye Tracking Leader Tobii Pro Choose EIZOs Monitors for Behavioral Research Solution. https://www.eizoglobal.com/solutions/casestudies/tobii-pro/ (Oct, 2022).

[9] Nasif Imtiaz, Akond Rahman, Effat Farhana, and Laurie Williams. Challenges with Responding to Static Analysis Tool Alerts. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 5 2019.

[10] Brittany Johnson, Rahul Pandita, Justin Smith, Denae Ford, Sarah Elder, Emerson Murphy-Hill, Sarah Heckman, and Caitlin Sadowski. A cross-tool communication study on program analysis tool notifications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, page 73–84, New York, NY, USA, 2016. Association for Computing Machinery.

[11] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *2013 35th International Conference on Software Engineering (ICSE)*, pages 672–681, 2013.

[12] Jenny T Liang, Chenyang Yang, and Brad A Myers. A large-scale survey on the usability of ai programming assistants: Successes and challenges. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE, 2024.

[13] Luke Nguyen-Hoan, Shayne Flint, and Ramesh Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, New York, NY, USA, 2010. Association for Computing Machinery.

[14] ChatGPT | OpenAI, 2024. Retrieved Feb 8, 2024 from `https://chat.openai.com/`.

[15] GitHub Copilot Your AI pair programmer, 2024. Retrieved Feb 8, 2024 from `https://copilot.github.com/`.

[16] Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, and Sven Apel. Correlates of programmer efficacy and their link to experience: a combined EEG and eye-tracking study. *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 11 2022.

[17] Caitlin Sadowski, Edward Aftandilian, Alex Eagle, Liam Miller-Cushon, and Ciera Jaspan. Lessons from building static analysis tools at Google. *Communications of The ACM*, 61(4):58–66, 3 2018.

[18] Caitlin Sadowski, Jeffrey van Gogh, Emma Söderberg, Ciera Jaspan, and Collin Winter. Tricorder: Building a program analysis ecosystem. In *2015*

IEEE/ACM 37th IEEE International Conference on Software Engineering ICSE 2015, pages 598–608, United States, 2015. IEEE - Institute of Electrical and Electronics Engineers Inc. 37th IEEE International Conference on Software Engineering, ICSE 2015 ; Conference date: 16-05-2015 Through 24-05-2015.

[19] William Saranpää, Felix Apell Skjutar, Johan Heander, Emma Söderberg, Diederick C. Niehorster, Olivia Mattsson, Hedda Klintskog, and Luke Church. Gander: a platform for exploration of gaze-driven assistance in code review. In *Proceedings of the 2023 Symposium on Eye Tracking Research and Applications*, ETRA '23, New York, NY, USA, 2023. Association for Computing Machinery.

[20] Del Siegle. Open, in vivo, axial, and selective coding, 6 2023. Retrieved Feb 8, 2024 from `https://researchbasics.education.uconn.edu/open-in-vivo-axial-and-selective-coding/`.

[21] Visit Sweden. Fika like a Swede, 2 2023. Retrieved Feb 8, 2024 from `https://visitsweden.com/what-to-do/food-drink/swedish-kitchen/all-about-swedish-fika/`.

[22] ThadT. Software Analysis (Basic Introduction). 9 2023. Retrieved Feb 8, 2024 from `https://medium.com/@thadt/software-analysis-basic-introduction-dea3a735503a`.

[23] Tobii. Eye tracking fully integrated and baked right into the very latest high performance gaming devices from alienware, acer and msi. `https://gaming.tobii.com/products/laptops/` (Oct 3, 2022).

[24] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C. Gall. Context is king: The developer perspective on the usage of static analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 38–49, 2018.

[25] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. How open source projects use static code analysis tools in continuous integration pipelines. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 334–344, 2017.