Designing A Multi-modal IDE with Developers: An Exploratory Study on Next-generation Programming Tool Assistance

Peng Kuang
Lund University
peng.kuang@cs.lth.se

Emma Söderberg
Lund University
emma.soderberg@cs.lth.se

Martin Höst
Malmö University
martin.host@mau.se

Abstract

Researchers have envisioned and pioneered data-driven programming assistance for developers based on their interaction with the tools via multiple sensors such as eye trackers, microphones, and AI. However, these new sensors gather sensitive data from programmers, to what extent users can accept them and in what form they may work well are largely unclear. Meanwhile, developer tools such as static analyzers have been criticized for poor usability and not involving end users enough during their development. To make data-driven programming assistance design work, toolmakers need to partner with programmers.

In this paper, we adopt a design process under the guidance of Participatory Design (PD) which aims to empower end users. Our design pipeline builds on a survey of 68 professional developers. The next stage is a design workshop with five participants sketching out ideas for alleviating the pain points reported from the survey. Based on these inputs, we then developed two types of tentative design representations consecutively – three conceptional designs and one low-fidelity prototype[1]. Lastly, we tested the interactive digital prototype with five experienced programmers.

From the user test, we learned that developers have more interest in gaze-driven assistance than voice-based assistance for reading and understanding code in an integrated development environment. Further, we report our hands-on experience with involving developers from the beginning to the end of this design process. This informs future work on using PD to study the development of developer tools.

1. Introduction

Data-driven techniques are making their way into developer tools. In the past decade, eye tracking technology has matured significantly and eye trackers are now cheaper and more available. Laptop manufacturers have started to embed eye trackers into their high-performance models for gamers (Tobii, 2023). Apple has introduced a three-dimensional interface with hands, eyes, and voice, alongside a bundle of productivity tools on its VisionOS platform (Apple, 2023), which makes reading natural language texts on a virtual screen foreseeable. We are also seeing early explorations of gaze-driven (Saranpää et al., 2023; Cheng, Wang, Shen, Chen, & Dey, 2022; Santos, 2021; Shakil, Lutteroth, & Weber, 2019; Radevski, Hata, & Matsumoto, 2016; Glücker, Raab, Echtler, & Wolff, 2014) and voice-based (Paudyal, Creed, Frutos-Pascual, & Williams, 2020; Talon, 2024) assistance in development tools. Meanwhile, we are in the middle of a shift toward AI programming (Liang, Yang, & Myers, 2024) or Autonomous Software Engineering (Zhang, Ruan, Fan, & Roychoudhury, 2024; Yang et al., 2024; AI, 2024), where Large Language Models (LLMs) trained on huge amounts of data are assisting developers with development. These data-driven technologies, gathering data from sensors or code bases, have the potential to enrich users' experience and improve their productivity.

We see an opportunity to incorporate these technologies into building the next generation data-driven developer tools, as some researchers have envisioned (Kuang, Söderberg, Niehorster, & Höst, 2023; McCabe, Söderberg, Church, & Kuang, 2022) and pioneered (Saranpää et al., 2023; Cheng et al., 2022; Hijazi et al., 2021). However, incorporating these new sensors implies gathering significantly more and possibly sensitive data from programmers. Previous work demonstrates that programmers may have concerns with such data collection (Kuang, Söderberg, & Höst, 2024). Further, how to encapsulate these fancy new technologies into a good design that can be accepted by end users can be a challenge.

[1]A replication package is available at: https://doi.org/10.5281/zenodo.13853909

Taking AI as an example, it did not gain wide user acceptance until OpenAI captured it into the design of ChatGPT. Exploratory gaze-based developer tools have primarily used eye-tracking for code navigation on the interaction level and information generation about developers' collaborative work such as code review. As for voices, it has so far mainly been explored for accessibility improvement, for instance, as an alternative input channel for developers with visual impairment (Paudyal et al., 2020). It seems the exploration in this field is still in its infancy. There are possibly other user scenarios and developer cohorts that may benefit from the integration of these emerging modalities as well. To make it work, we believe developer tools need to be designed carefully, and the design process needs to partner with programmers.

How do we involve developers in the design process? As an example, the community around Open Source Software (OSS) is driven by programmers and there are many examples of successful projects (e.g., Linux, Apache, and Mozilla Firefox). However, many OSS projects have been criticized for poor usability or not being end-user friendly (Hellman, Cheng, & Guo, 2021). It is believed that developers tend to adopt a code-centric mindset and that this may create a gap between a human-centric designer role and a developer role, along with the different training and skills associated with these two roles (Maudet, Leiva, Beaudouin-Lafon, & Mackay, 2017). Even though developers who work with developer tools seem to embody the user, the designer, and the developer, there are proven records that the tools they make can still be poor to use because of this gap. To mitigate such a risk, we believe it is necessary to involve users other than the toolmaker developers themselves in the design process. We need a design process that reduces this gap.

Participatory Design (PD) emerged from Scandinavia to empower the workers to balance or counter the insights of management in the workplace (Bannon, Bardzell, & Bødker, 2018; Schuler & Namioka, 1993; Ehn & Sandberg, 1979). PD roots in a strong commitment to democracy (Bødker, Dindler, & Iversen, 2022), which can be translated into the paramount care for end users. It further pronounces that the designer needs to partner with users in the design process to discover possibilities or alternatives for a technology being imposed on them (Bødker et al., 2022). As an outcome of such a design process, *"mutual learning"* for both sides is curated. What distinguishes PD from other design methods is that it takes both a moral position and a pragmatic position (Carroll & Rosson, 2007) and that it emphasizes *"democratic empowerment"* but not just *"functional empowerment"* (Clement, 1996). We think PD may be a good fit for designing next generation data-driven programming tools with programmers.

In this paper, we explore participatory design through a series of design activities with professional developers to rapidly prototype and test out data-driven programming tools powered by AI and other assistive sensory technologies, e.g., eye tracking, that are novel to the current typical software development environments. This work is in line with what Kuang et al. (Kuang et al., 2023) have proposed as a method to study gaze-driven assistance. We focus on investigating the RQ: *What do developers want the next generation data-driven programming assistance to look like?* Through the design pipeline, we gradually capture developers' needs and wants with the next-generation programming tool assistance via several intermediate design representations and eventually actualize it into a tangible multi-modal Integrated Development Environment (IDE). The IDE contains four features: two gaze-driven, one voice-based, and one AI-enabled. We have evaluated this interactive artifact with experienced programmers and learned that there is greater interest from developers in gaze-driven assistance than voice-based assistance in the case of reading and understanding code in an IDE.

This exploratory study reports our hands-on experience on how to practice participatory design in partnership with developers for developing new types of programming tool assistance. Our findings, through the *"mutual learning"* between the designer and developers, provide insights into what modality may work and not work well with developers for designing the next-generation programming assistance. We hope that our reported experience, of end-to-end involvement of programmers throughout the PD process, will aid other toolmakers interested in exploring this direction.

2. Related Work

In this section, we give a brief overview of related work in multi-modal developer tools and the use of participatory design in the development of developer tools.

2.1. Multi-modal Developer Tools

A multi-modal interface usually supports the interaction means of gaze and voice, apart from the conventional input means of keystroke, mouse, and touch through one's hands (Benoit, Martin, Pelachaud, Schomaker, & Suhm, 2000). Owing to the naturalness that it resembles human-human communication and the expressiveness and adaptability that it provides (Oviatt & Cohen, 2000), multi-modal interface has attracted significant interest from HCI researchers in recent decades. Especially in the current era of AI, people are even more passionately envisioning and actualizing intelligent systems, also for developer tools.

Since reading code constitutes a significant part of developers' work, some researchers investigated the feasibility of utilizing gaze for several software development tasks that were undermined by code reading. Several studies have examined its suitability for code navigation (Santos, 2021; Shakil et al., 2019; Glücker et al., 2014), while several others experimented with it for code review (Saranpää et al., 2023; Cheng et al., 2022; Hijazi et al., 2021). The overall gaze behavior during software development has also been inspected (Clark & Sharif, 2017). The modality of voice has been less studied but has been piloted in the context of empowering developers with disabilities (e.g., visual (Paudyal et al., 2020) or hand impairment (Talon, 2024)).

The above-mentioned studies share a human-factor motivation and usually contain a component of a user study. However, they primarily focus on the introduction of implemented techniques, and the user studies are almost all done post-implementation for testing the usability of the tools. As software development is a high-cost activity, we think it is worth involving users from an early stage of the design process to mitigate the risk of having (if not completely avoiding) implementations that do not fit into users' needs. Further, the multi-modal features presented in our design also cater to user scenarios different from the ones investigated by the previous studies.

2.2. Participatory Design and Developer Tools

Participatory design is a well-known design method to software engineering researchers and gained significant interest throughout the 1990s to 2010s. It is usually brought up along with user-centered design when researchers discuss research questions pronouncing user involvement or end-user software engineering.

Common developer tools emerge from the industry, the open-source community, and the scientific community. These three sources are not mutually exclusive as both companies and scientists can be members of the OSS community. To provide a structure for reporting the use of PD in developing developer tools, we group the related works into these three categories.

In the industrial context, some researchers (B. Johnson, Song, Murphy-Hill, & Bowdidge, 2013) have adopted PD for eliciting non-verbal inputs from target users to improve the design of static analysis tools. It seems that PD is used at surface level in this case or a conventional user study alike.

In the area of OSS, we did not find a study in the case of developing a developer tool explicitly saying that they have used PD (this does not imply our search is exhaustive) although some well-known developer tools such as Apache Maven are OSS. However, there are literature reviews (e.g., (Hull, 2021)) and many case studies (e.g., (Hellman et al., 2021; Wubishet, Bygstad, & Tsiavos, 2013; Iivari, 2009; Gumm, Janneck, & Finck, 2006)) on user participation in other types of computer systems that can inform and perhaps extend to developer tools. These studies found that there are both technical and social barriers (e.g., technical capability and social credibility (Mockus, Fielding, & Herbsleb, 2002), documentation (Hull, 2021), subculture literacy (Iivari, 2009)) for developer users (who are usually the first batch of users of OSS) to participate in and contribute to OSS development.

In scientific programming, a case study (Letondal & Mackay, 2004) shows that because the resources

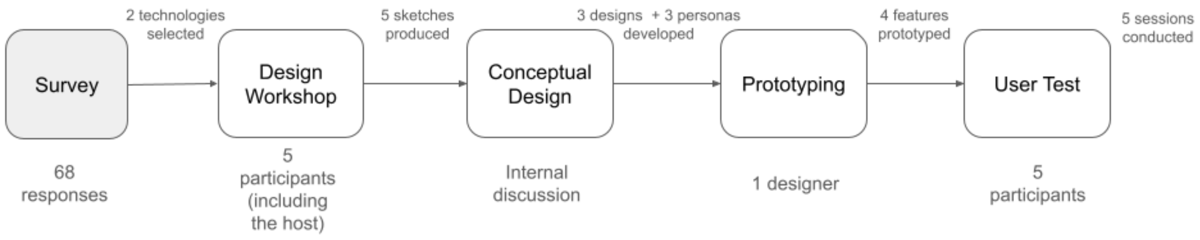


Figure 1 – Overview of the method. The greyed box means the component is not directly reported in this paper. It has been published separately but was done as a part of this study to inform the remaining components of the design pipeline.

for new hiring are limited, developers working with scientific teams employed PD as a design method to develop some intermediary tools to enable end-user scientists to perform some frequent but not-so-difficult programming tasks. However, such tools are not the typical developer tools used by professional developers for software development but are similar to business information systems.

To sum up, HCI methods (e.g., rapid prototyping, usability evaluation) are valued by researchers and have been adopted as a practice in the industry (Myers, Ko, LaToza, & Yoon, 2016) for developing programming tools. However, the explicit use of PD for such purposes seems to be rare, either in the industry, the OSS community, or the scientific programming community.

3. Method

The design process consists of a survey with professional developers, a design workshop, conceptual design, prototyping, and the user test (Figure 1). We conducted a survey targeting professional developers and received valid responses from 68 participants. In the workshop, we presented the pain points collected from the survey for participants to design solutions for. To catalyze brainstorming and sketching, we demonstrated a gaze-driven code review tool (Saranpää et al., 2023) as a data-driven programming assistance example. We then used the outputs curated from the previous modules to develop conceptual design ideas and create personas representing different types of prospective users. After that, we translated these design ideas into four features of a low-fidelity prototype. Lastly, we tested the interactive prototype with five prospective users who are experienced programmers.

3.1. Survey

The survey received responses from 68 professional developers located in 12 countries. In the survey, we asked developers about their experiences with current programming tool assistance and their perceptions of programming tool assistance in the future. The survey (Kuang et al., 2024) has been published separately and its preliminary results were leveraged in planning the design workshop.

3.2. Design Workshop

In the survey, there was a checkbox question for participants to indicate their interest in participating in the workshop. 12 participants ticked the checkbox and provided their email address. We reached out to these twelve potential participants who expressed their interest via email to poll their availability in the upcoming weeks. Four of these potential participants accepted the invitation. Because there was no single time slot that could accommodate all the participants, we chose the one that seemed to suit the majority. Since some of these participants were slow with the response, we actively recruited some other participants in person through our social circle to reduce the risk of no turn-out. Eventually, only one from the online participant pool managed to attend the workshop due to their challenging availability or large time difference. The rest were from the convenient recruitment in-person.

We organized a hybrid design workshop session, with one participant online and others onsite. Two participants were Ph.D. students in Sweden but in different cities. One was in the field of Computer

Science and was familiar with specialized developer tools such as program analysis. The other had a background in Electrical Engineering and worked with wearable devices and machine learning (ML). Two other participants were research assistants who have been working as developers for two years in the Department of Computer Science at Lund University. The host (a.k.a. the first author) also joined and was counted as the fifth participant since we adopted a co-design formality. All participants were given access to a digital collaborative tool called Box Canvas. The participants onsite were also provided with markers and white papers. Participants were encouraged to choose whichever way they felt most comfortable to output their design ideas.

The workshop followed the procedure of introduction, a demo of a screen-based eye-tracker, as well as a code review tool with integrated gaze-analysis (Saranpää et al., 2023), brainstorming, and debriefing. We included these two demos to lower the barrier for participants (as we are informed from the survey that many developers do not know what a contemporary eye tracker looks like and how eye-tracking has been experimented with software development) and to elicit ideas around it. However, since the participants were new to this form of collaboration, we did not prescribe that the focus had to be eye-tracking to get the best out of the workshop. Instead, we presented a list of pain points (e.g., code comprehension, tool setup, dependency management) and we asked participants to select one or two pain points as the problem of their interest. We asked them to brainstorm ideas to address the selected pain points using but not limited to machine learning or eye-tracking.

The pain points we presented were those reported by the developers from the previously mentioned survey. We suggested machine learning and eye-tracking as the potential underlying technologies for the designs because the developers in the survey were positive toward the former and neutral toward the latter. Other candidate technologies such as gamification toward which the developers from our survey were negative were dropped.

3.3. Conceptual Design

Following up on the design workshop, the first author developed three conceptual designs in the form of wireframes (Guilizzoni, n.d.). According to Johnson and Henderson, a conceptual design is a high-level model that describes the major design metaphor or analogy (J. Johnson & Henderson, 2002). We also developed personas (Rogers, Sharp, & Preece, 2011) to characterize prospective users. As Pruitt stated (Pruitt & Grudin, 2003), persona suits early-stage PD very well as it can unearth sociopolitical issues and is a great complement to other scenario-based usability methods. Our designs focus on leveraging eye-tracking, speech/voice, and machine learning as a kind of support to alleviate the key pain point – code comprehension that has been brought up during the design workshop for developers.

3.4. Prototyping

After internally discussing the designs within the author group, and with some colleagues with expertise in interaction design, the first author created a low-fidelity prototype using the popular design tool Figma (Figma, n.d.). According to Rogers et al. (Rogers et al., 2011), prototyping is a cheap way for designers to actualize design ideas and conduct user tests to select from design options. The first and second authors approached a few target users through their networks, some of these target users were the same persons as those who had expressed an interest in attending the earlier design workshop but missed it due to availability issues at the time.

3.5. User Test

The first author conducted user tests with five participants (referred to as PT1-PT5 later) via Zoom. Three are PhD students from the same university as the first author, all with significant industrial programming experience, and two external PhD students from a different university in Germany, with an academic background in researching Human-Computer Interaction of programming tools. None of the participants had participated in the earlier design workshop. We chose Zoom to conduct the user test even with the participants who work at the same university as the first author because they may not be physically in the same space on the same day and to capture their screens and the interviews. The first author video-recorded the test sessions with the users' consent.

The user tests followed the process of introduction, think-aloud testing, a semi-structured short interview, and debriefing. The five users conducted the user tests separately with the first author. The sessions started with the background of the study and instructions on how to adjust the size of the interface of the online Figma prototype for later use, the remaining steps were carried out as outlined below. After the user tests the recorded material was analyzed, also described below.

3.5.1. Think-aloud Testing

After that, the users were asked to explore the prototype think-aloud with the first author observing. The first two participants were not given a specific task, because the first author was interested in learning about how they would discover the entries of the features. This was to partly mimic what would happen after a new icon, for a new functionality, was introduced into a complex IDE. However, this caused some disorientation for these two participants, so the first author prescribed a task of reading and understanding the code for the later participants.

3.5.2. Semi-structured Interview

Next, the users were interviewed by the first author with structured questions as follows:

1. How would you describe your overall experience of using this prototype?
2. What did you like and dislike about the prototype?
3. Are there any parts confusing to you?
4. What parts would you change or remove?
5. Do you have any comments, particularly for any part of the prototype or the session?

The interviewer probed with spontaneous questions wherever there was an interest in further investigation or clarification.

3.5.3. Debriefing

Finally, the first author debriefed the user test session and prototype design to the participants. This was especially needed for some of the participants who were new to such concepts or study methods and artifacts. This also aligned with the *"mutual learning"* between the users and the designer that participatory design promotes. The sessions lasted for 25 to 45 minutes.

3.5.4. Data Analysis

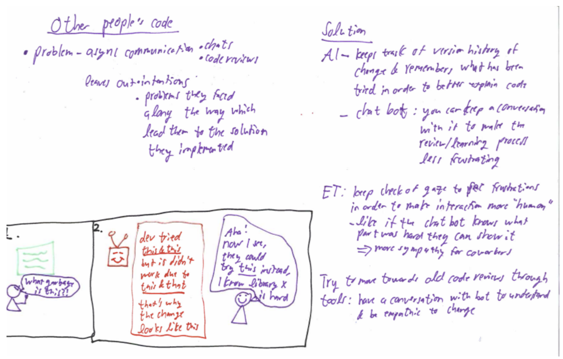
For the interviews during the user tests, we used Microsoft Word's premium feature of auto-transcribing to generate the transcripts. The first author, who was also the interviewer with the users, skimmed each of them and marked the lines that were related to each of the features of the prototype. Based on this, the first author summarized the points for each feature. And when there was anything unclear, the first author re-consulted the videos.

3.6. Threats to Validity

We identify the following threats in our study:

Construct validity: For the design exploration, our approach may not authentically comply with how it is prescribed to be done in the textbook. This is partly because participatory design is a rich theory and overlaps with many neighboring design methods such as user-centered design, collaborative design, and interaction design. It is a learning process for us to better understand it by practicing it. There are also some components we adapted for the sake of participants or the process, e.g., we deliberately used "co-design" instead of "participatory design" to communicate with our participants. In our view, co-design is a simpler term and more self-explanatory. It conveys better the partnership and the action required to our participants who do not necessarily know the design theories. As co-design is commonly viewed as a branch of or an interchangeable, modern term for participatory design (Wikipedia, 2024), we deem this impact minimal. Further, as the original contributors of participatory design stated, there is no uniform way to practice this method (Schuler & Namioka, 1993).

Internal validity: The user test data was auto-transcribed so a complete coverage of the conversations was ensured. Although the second and third authors did not read the transcripts themselves, the first



Other people's code

• problem - async communication • chats • code reviews

leaves out • intentions
• problems they faced along the way which lead them to the solution they implemented

Solution

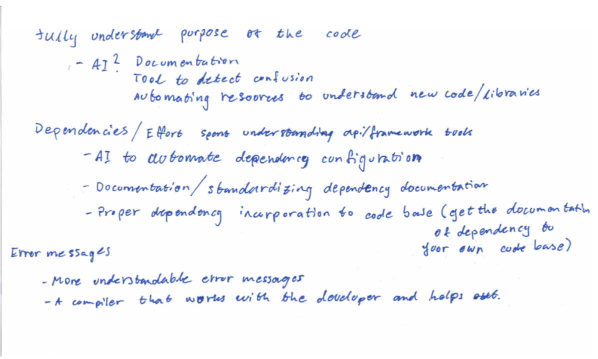
AI - keeps track of version history of change & remembers what has been tried in order to better explain code
- chat bot: you can keep a conversation with it to make the review/learning process less frustrating

ET: keep check of gaze to ~~[illegible]~~ frustrations in order to make interaction more "human"
- like if the chat bot knows what part was hard they can show it
⇒ more sympathy for coworkers

Try to move towards old code reviews through tools: have a conversation with bot to understand & be empathic to change

1.
What garbage is this?!!
2.
dev tried this & this but it didn't work due to this & that
that's why the change looks like this
Aha! now I see, they could try this instead, I know library X is hard

(a) Hard-copy sketch by participant PW1.



fully understand purpose of the code
- AI? Documentation
Tool to detect confusion
Automating resources to understand new code/libraries

Dependencies / Effort spent understanding api/framework tools
- AI to automate dependency configuration
- Documentation / standardizing dependency documentation
- Proper dependency incorporation to code base (get the documentation of dependency to your own code base)

Error messages
- More understandable error messages
- A compiler that works with the developer and helps out.

(b) Hard-copy sketch by participant PW4.

Figure 2 – Design workshop output: Sketches from PW1 and PW4.

author located the corresponding paragraphs through keyword-matching and marked them to allow the other two authors to cross-check when writing. Also, because the first author was the one who conducted the interviews with the participants during the user test sessions, the hands-on experience helps reduce the possibility of misinterpretation of the data.

External validity: For the design workshop and user test, our participants lean toward an academic profile. This is because of the challenging availability of industrial practitioners and the timing of being close to big holidays. The homogeneity of this aspect may bias the design outputs and findings from the user test. Hence, their generalizability shall be interpreted with caution.

4. Results

We present the results from the design workshop, conceptual design, prototyping and user test. We refer to participants in the workshop with the prefix "PW" and participants in the user test with the prefix "PT".

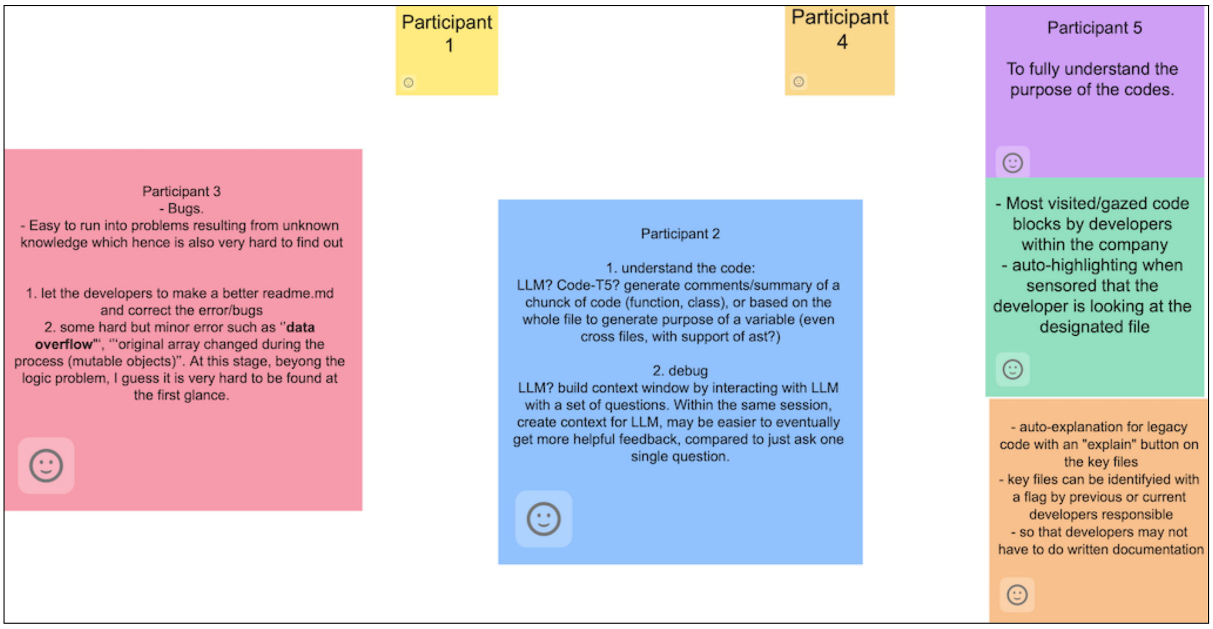
4.1. Design Workshop

Four out of the five participants picked a pain point related to code comprehension, especially understanding the code written by others. As shown in Figure 2a, participant PW1 mentioned the scenario of reviewing others' code, proposing an AI-enabled, interactive chatbot that allows consecutive conversations with the programmer to better explain code with richer context information, e.g., the history of changes and what has been attempted. Participant PW1 also touched upon tracking programmers' gaze to pinpoint frustrations so as to introduce more *"human"* support. Participant PW4 (Figure 2b) also selected the problem of *"fully understand purpose & the code"*. The participant considered AI as a viable solution to *"detect confusion"* of programmers and to *"automating resources"* to facilitate the understanding of new code or libraries.

According to Figure 3, we can see participant PW2 also mentioned utilizing Large Language Models to generate summaries or explanations for a code base or variables of interest and to retain context within a time frame for debugging queries. PW3 wished for better support for finding bugs caused by unknown knowledge which therefore is very difficult to fix. He verbally elaborated on an example that was associated with changed dimensions and values of matrices during his composition of machine learning models. The host PW5 also picked a problem related to code comprehension and suggested several ways of providing gaze- and voice-based support to improve it. Participant PW3 reacted very positively to the voice-based proposal.

4.2. Conceptual Design

Consolidating the ideas and discussion inputs from the design workshop, the first author translated a selection of elements from the design workshop into three main conceptual designs, while taking into account the feasibility of prototyping and implementation as well as finding a balance between truly good


Participant 1

Participant 4

Participant 5

To fully understand the purpose of the codes.

- Most visited/gazed code blocks by developers within the company
- auto-highlighting when sensored that the developer is looking at the designated file

Participant 3
- Bugs.
- Easy to run into problems resulting from unknown knowledge which hence is also very hard to find out

1. let the developers to make a better readme.md and correct the error/bugs
2. some hard but minor error such as ''**data overflow"**', '''original array changed during the process (mutable objects)". At this stage, beyong the logic problem, I guess it is very hard to be found at the first glance.

Participant 2

1. understand the code:
LLM? Code-T5? generate comments/summary of a chunck of code (function, class), or based on the whole file to generate purpose of a variable (even cross files, with support of ast?)

2. debug
LLM? build context window by interacting with LLM with a set of questions. Within the same session, create context for LLM, may be easier to eventually get more helpful feedback, compared to just ask one single question.

- auto-explanation for legacy code with an "explain" button on the key files
- key files can be identifyied with a flag by previous or current developers responsible
- so that developers may not have to do written documentation

Figure 3 – Design workshop output: Digital post-it notes on Box Canvas by participants PW2, PW3, and the host PW5.

ideas and the research focus. The three conceptional designs are an individual view of the developer's gaze with personal work notes, a collective view of a developer team's gaze, and a conversational code explainer empowered by ML.

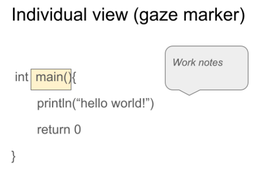
- The individual view (Figure 4a) mimics a text marker, which auto-highlights the code fragments for a developer. These code fragments will be based on the developer's gaze metrics, e.g., the top 3 code fragments with the longest dwell time. We also added a component of work notes to this personalized space.
- The collective view (Figure 4b) metaphorizes a mirror that reflects or synthesizes a developer team's gaze behavior on a high level. For instance, for each file of a huge program, it will show the top 5 code fragments that have been fixated by the team collectively for the longest time.
- The code explainer (Figure 4c) is conceptualized as similar to a tour guide working in a museum, who is knowledgeable about all the items, e.g., historical artworks, to give as much information as needed to the queries from visitors a.k.a. developers.

Lastly, we also developed personas (Figure 5) to capture the prominent profiles among the developers we observed from the survey and design workshop. The three personas represent seasoned professional developers, scientific programmers, and early-career junior developers, respectively.

4.3. User Test

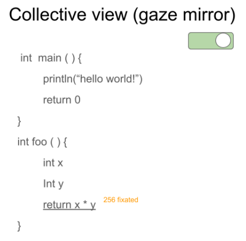
The conceptual designs were translated into a low-fidelity prototype with four corresponding features. Two features are related to gaze, one to voice, and one to AI. The individual and collective views were converted into the gaze marker and gaze mirror. The voice notes resemble the work notes. The code explainer becomes the conversational AI assistant. The concept of work notes was actualized primarily in the voice notes feature but also partially as the filtered history of commands used in the AI assistant feature. The latter design echoes the finding from a survey study that developers often use ChatGPT for syntax recall (Liang et al., 2024). A demo of the prototype can be viewed here: https://youtu.be/J9cGrK4oZ5U

We recruited five experienced programmers to participate in the user test. Each of them tested the prototype in a think-aloud manner separately with the first author observing.



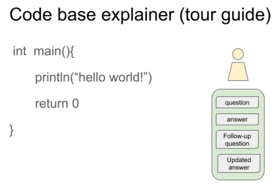
```
int main(){
    println("hello world!")
    return 0
}
```

(a) Individual view of a developer's gaze and work note.



```
int main ( ) {
    println("hello world!")
    return 0
}
int foo ( ) {
    int x
    Int y
    return x * y
}
```

(b) Collective view of a team's gaze.



```
int main(){
    println("hello world!")
    return 0
}
```

(c) Code base explainer.

Figure 4 – Conceptual designs.

Ahmed

A senior software engineer with 15 years of experience in programming in C & C++ but now is switching to Rust.

Self-taught in programming and developed his career in the gaming industry.

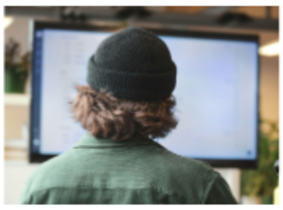
Don't like new tools to be added as another layer to his already-complex tool chain.Prefer simple print debugging method and textual & CLI-based tools if he ever needs any.

Beibei

A first-year PhD student who did both her bachelor & master in CS.

Specialized in ML-related research.

Want to have some tool to help her debug ML models, especially for the case where only data in the matrices were modified while the shapes remained the same.



Karl

A junior software developer in a start-up company. This is his first full-time job and still in his 6-month probation period.

Did his bachelor in software engineering.

Want to have some tool to help him quickly understand the code base he needs to work with and deliver his first merge request to prove his capability to his supervisor.

Figure 5 – Conceptual design output: Personas.

4.3.1. Individual View and Collective View

For the two gaze-related features (Figure 6a and Figure 6b), one out of the five users showed an evident interest in them, while three indicated moderate interest in seeing alternatives. Although the participants may not agree with the current visualization and/or the selection of the gaze data, they demonstrated some extent of acceptance of the leverage of gaze data in a programming environment or certain software development scenarios. Participant PT1 and PT2 somewhat agreed with its potential of being useful in certain cases specifically tailored to them, while participant PT3 disagreed with the usefulness of leveraging gaze data in the design. They explained:

> **Participant PT1**: *"I think I would be interested to see anyone's trace just to get an understanding of what the data looks like right so."*

> **Participant PT2**: *"...but maybe if you're like in a big project, there's like some code that never gets any love because nobody's looking at, maybe that would make sense trying to find stuff that nobody's looking at."*

> **Participant PT3**: *"I don't think it's that useful, because yes, I probably looked at this line*

of code the most for like three seconds or five seconds, but do I actually have to care about that? Why should I care?"

In summary, participants perceived these two features as somewhat interesting in the way they reveal programmers' gaze behavior but had different opinions on what way it should reveal the gaze and to what extent the gaze should be revealed to be recognized as useful. Participant PT1 was very interested in seeing others' individual gaze traces and defining their gaze traces for didactic purposes. Participant PT2 thought that locating the least gazed code of a file would be more informative. Participant PT4 suggested it would be more interesting to mark the code that their co-workers looked but they missed out. Participant PT5 was also interested in seeing others' gaze traces but only of those of their interest.

4.3.2. Voice-based Work Notes

For the feature of voice notes (Figure 6c), it was received worse although some participants praised this idea in the workshop and on another occasion. When it was presented as a tangible feature in a prototype, it was less favored. Some factors were attributed to the limitations of the mock-up design itself such as limited interactiveness and lack of accuracy. But primarily it was because the participants were not convinced of the value it was proposing - to be an alternative for text comments or written documentation. Participants had a very critical and practical eye when examining it as a concrete feature of the prototype.

Participant PT3: *"But then regarding the voice note, I think theoretically it provides an alternative of not looking at the code by just hearing some voice explanation or summary of the code. But I mean there will be some obstacles of really using this feature, because first of all, why are you looking at the code? You do not bother the other colleagues or team members in the same office, but if you want to play voice notes then you probably need to plug in your headset."*

This kind of issue was escalated to how the voice notes get updated if the developer has changed the code, what will happen if the code is pushed to a remote repository, and so on. These all are rather reasonable and practical considerations. Nonetheless, participants PT2 and PT5 still saw its suitability for some rarer cases. Participant PT2 stated the potential of using it to record *"meta-commentary"* for the students instead of inserting *"a giant comment in the middle of code"*. Participant PT5 pointed out that this kind of feature may be an enabler for *"dyslexic"* users or non-native speakers who prefer listening to audio over reading texts for learning. We also proposed the user scenarios: when developers reach fatigue from reading code, they can switch to this alternative; when their co-workers, e.g., who is the original author of a critical method, are unavailable for consulting, this can be of help.

4.3.3. Conversational AI Code Assistant

Lastly, for the conversational AI-enabled code assistant (Figure 6d), the fact that the choice of example conversation lacked a connection with the code presented in the editor led to some users' strong dissatisfaction with this feature.

Participant PT4: *"I mean, this seems completely disconnected. It shows terminal commands, it has nothing to do with the code. It says it's a code assistant but it's like my bash history excerpts."*

The same perception of lack of relevance and thus low-value or valueless was shared by participants PT3 and PT5. Participant PT5 stated,

Participant PT5: *"I don't really see how the assistant fits into the other features and I think the product as such would be leaner without that."*

While participant PT3 held the same opinion as PT5 in the beginning, he shifted his attitude when the interviewer probed with improvement or alternative ideas and he believed in the potential usefulness of those propositions.

The participant's view pivoted when a new design was proposed, for instance, participant PT3 reflected as follows:

> **Participant PT3**: *"The last feature I mean, if it's just general code assistant, they're giving some hints like, you know, when you usually when we usually start an idea like IntelliJ or Eclipse. It will prompt up. It will pop up with some general...programming hints or some short shortcuts that you might commonly use, but this kind of assistance [is] very general. It has nothing to do with the code."*

When asked if an added action that would analyze the current code snippet or summarize the current file, would be interesting, participant PT3 responded:

> **Participant PT3**: *"Yeah, definitely... I mean, that is something that I always looking at ... For example, if you review any code pretty much you don't directly get into the code, you will read the comment to the Java doc...the documentation for it first, try to understand the what is the intention of this method. Or what they say, [the] intention of this class. So regarding the particular method, what...the input value [and] output value in which format? So I mean you have to get some kind of general sense what this class or this method is for and if that information is provided by this code assistant, I would consider it's useful."*

Participants PT1 and PT2 were more open to this feature in part merely because of the concept of having an interactive code assistant. This might be influenced by their academic background and research interest in human-computer interaction.

In addition, the filtered history of commands used also triggered feedback from some participants. PT1 and PT2 acknowledged the usefulness of the most frequently used commands. PT1 further commented that rarely used commands might be more useful, as users tend to forget them due to their infrequent use. On the other hand, PT4 did not think it would be very useful and suggested this problem could be resolved by searching one's command records, for example, on the Linux command line interface. This part of the AI assistant feature did not receive any particular comments from the remaining two participants.

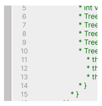
Summary

The design process comprised a pipeline of different components each with its low-level focus and staged goal. However, they curated learning together for the designers to better understand the user needs and to construct or modify the design representations. On the other hand, the users also became more aware of what the proposed and designed assistance would look like through the designers' idea pitches and their hands-on experiences interacting with the tangible artifacts.

Specifically, from the user test, we found participants are interested in the gaze-related features because of the novelty of the underlying technology, the interest in inspecting self's and others' gaze and their deviations, and the potential they see that it can be applied to other scenarios such as pedagogical and accessibility use cases. The voice notes and AI code assistant received less interest because some participants were concerned with the practical use (the former) or distracted by the surface-level usability weaknesses (the latter) of the feature.

5. Discussion

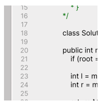
In this section, we draw on the high-level results to revisit the RQ. We examine the connection between our study and related work. We also reflect on the method and share the future work that we contemplated.



```
        * int v
        * Tree
        * Tree
        * Tree
        * Tree
        * Tree
          * th
          * th
          * th
        * }
      * }
```

(a) Prototype feature 1: individual view of developer's gaze (gaze marker).

(b) Prototype feature 2: collective view of a developer team's gaze (gaze mirror).



```
      * }
    */

    class Solut

    public int r
      if (root =

      int l = m
      int r = m

```

(c) Prototype feature 3: voice explanation (tour guide).

(d) Prototype feature 4: AI code assistant (tour guide & personalized work notes).

Figure 6 – Prototyping output: Screenshots of the four features of the prototype.

The answer to the RQ "What do developers want the next-gen programming assistance to look like?" is multi-layered. The layers correspond to the components of the design pipeline. From the survey, we learned that developers are positive toward AI and unsure about eye-tracking as the enabling technology for future programming tool assistance. For the novel sensor eye-tracking, they did express privacy concerns. During the design workshop, participants exhibited varying degrees of motivation or interest in utilizing AI/ML, gaze, and voice to sketch solutions for the pain points that the current programming tools seemingly failed to address. These selections were later conceptualized as different functionalities alleviating a common pain point – code comprehension. Incorporating these functionalities gave birth to a prototype of a multi-modal IDE which reflects the designers' interpretation of what the developers desired.

From the user tests with this prototype, we learned that experienced programmers want different programming tool assistance, but share a common perception that the design must be of practical use in helping them understand code more efficiently or effectively. Although the designs of the features that we prototyped are different from what we saw in related works, they did not gain particular favor from experienced programmers. We believe this implies that the underlying technology may have the potential to support developers, but finding the best shape of design encapsulating it may be challenging. However, compared with the voice-based feature, the gaze-based features appeared to have attracted more interest from the participants. This may be due to the efficiency of producing and consuming voice content. For instance, the effort needed to design the content for a useful voice note and to organize it well verbally (e.g., in terms of fluency, clarity, and tone variation) may surpass the effort needed for writing the documentation in some cases. Furthermore, although the question about privacy was not directly asked during the design workshop and user test, the participants never brought up the concerns either. There seems to be a discrepancy between the concerns written in the survey and the ones expressed in the hands-on and face-to-face sessions. This seems to resemble what researchers said about consumers on using AI (Siau & Wang, 2020). That is, users may “pay lip service” to privacy issues but are pragmatic in their behavior.

5.1. Reflection on the Method in the Study

The design workshop is one of the most challenging parts of this study. Recruitment of human participants is usually expensive. We assume this is partly attributed to the limited availability of practitioners during work days. While filling out a survey takes 10 to 15 minutes, attending a workshop with active involvement is more demanding and usually requires 45 to 60 minutes. Further, we speculate that participation in a research study during a workday might be perceived as unethical by some practitioners who work in the industry if not in the context that the researchers are collaborating with their employers. This explains that the participants who expressed interest in attending and made it to attend our workshop were mostly from academia.

For the user tests, our participants have an academic background or a background of having recently shifted their career from industry to academia. With capable experienced programmers, we advise researchers to use as realistic code as possible and with reasonable complexity and cognitive load. Educational code snippets such as data structure manipulation and algorithms may still be perceived as simple by some practitioners. In addition, we recommend giving participants a concrete task that they are used to or can relate to such as finding errors/bugs in the code even though the main goal is to test out the usefulness and usability of the design. This together elevates the realism of the user scenario and immersion of the task which in turn helps elicit more realistic reactions from participants and thus more useful data. Researchers also need to take care of the link between the features designed and the code presented. Participants tend to expect some coherency between them rather than treating them separately. Letting participants use the features with a connection to the code presented can avoid distracting them from the main task.

5.2. Directions for Future Research

For future work, we plan to select gaze-driven assistance as the primary feature for iterations and refinements as it receives the most interest and positive reactions or beliefs from participants in the user test sessions. We will revise the design of the gaze feature with the inputs and implement a high-fidelity prototype, either in the form of a custom IDE/code editor or a plugin published in one of the mainstream IDEs.

Reflecting on the design process, we learned that the junior developer persona (which shares characteristics with novice programmers to a large extent) may be the most beneficial programmer cohort for us to work with. First, we think experienced programmers or experts tend to have well-established programming tool preferences, e.g., a specific tool or debugging method such as the simple but effective print statement. Some of such programmers involved in our design process demonstrated a more critical and skeptical attitude toward the new enabling technologies that we proposed (it is even more prominent in the survey component/study (Kuang et al., 2024)). This implies that perhaps they are less enthusiastic about or open to novel programming tool assistance. Additionally, they are deemed to be rather resourceful and capable of helping themselves. Hence, there is less room for this type of design to be useful to them.

Second, with the scientific programmer persona, we observe the problems that they deal with are inclined to be data- and ML-centric. This demands domain-specific tool support that embeds deep knowledge of such problems. We see our design as a less fit for this goal as we envisioned it to be independent of the actual functionality of the code, that is, to be at the presentation layer but not the logic layer of the code. Lastly, we believe there are scenarios where early-career junior developers and prospective professional developers such as novice programmers are outstandingly goal-driven to deliver the results, for instance, to push the first pull request in their new job or get the group assignment done on time. They are potentially more open to diverse forms of support, especially given the fact that how they learn programming is drastically different from early generations of programmers, e.g., via ChatGPT or Co-pilot nowadays. In particular, we want to assist them with reading and understanding a code base for the first time. Studies (Green et al., 2023) report that it can take new software engineers 3 to 5 months to familiarize themselves with a new code base to be productive. Similar challenges may surface for prospective professional developers when collaborating on pre-scaffolded group assignments and contributing to open-source projects.

We further want to explicitly evaluate the PD method in a systematic way as per the recommendations from a survey on the use of PD (Bossen, Dindler, & Iversen, 2016) and a comprehensive review of this method in a related field (Spinuzzi, 2005). We wish to derive some representative quantitative criteria with developers to triangulate the benefits and gains of the use of PD, together with the qualitative data that we have partly reported in this paper. Lastly, we will also keep an eye on whether there is room for adopting AI/ML to enhance the gaze-driven assistance that we are going to realize.

6. Conclusions

In conclusion, we followed a design process that involved developers from the beginning to the end under the guidance of Participatory Design. We first surveyed professional developers about the pain points in their work and their attitude toward new technologies. Next, we organized a design workshop with five participants (including the host) to brainstorm and sketch out what programming tool assistance they wanted with the enabling technologies AI/ML and eye-tracking (toward which the developers from our survey have indicated positive or neutral attitudes). We then translated these inputs into three conceptual designs and developed three personas to capture the profiles that emerged from the previous modules. We further prototyped these designs as four features of a low-fidelity, multi-modal IDE. Finally, we tested this digital, interactive prototype with five experienced programmers.

From the discussions and interviews with these prospective users, we found that developers are open to new types of assistance powered by new and novel technologies. However, for them to be useful, their assistance must increase developers' efficiency or productivity. More specifically, developers from the user test recognize greater potential in gaze-driven assistance than in voice-based assistance facilitating code comprehension in an IDE.

7. Acknowledgements

We thank all the participants who took part in any component of the design process. We thank them for their curiosity, enthusiasm, creativity, and insights.

We 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.

8. References

AI, C. (2024, 3). *SWE-bench technical report.* (Retrieved 4 Apr, 2024 from `https://www.cognition-labs.com/post/swe-bench-technical-report`)

Apple. (2023, 10). *The Home View on Apple Vision Pro.* Retrieved from `https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro/`

Bannon, L., Bardzell, J., & Bødker, S. (2018, feb). Introduction: Reimagining participatory design—emerging voices. *ACM Trans. Comput.-Hum. Interact.*, *25*(1). Retrieved from `https://doi.org/10.1145/3177794` doi: 10.1145/3177794

Benoit, C., Martin, J.-C., Pelachaud, C., Schomaker, L., & Suhm, B. (2000). Audio-visual and multimodal speech systems. *Handbook of Standards and Resources for Spoken Language Systems-Supplement*, *500*, 1–95.

Bossen, C., Dindler, C., & Iversen, O. S. (2016). Evaluation in participatory design: a literature survey. In *Proceedings of the 14th participatory design conference: Full papers - volume 1*

(p. 151–160). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2940299.2940303` doi: 10.1145/2940299.2940303

Bødker, S., Dindler, C., & Iversen, O. S. (2022). *Participatory design.* Springer Nature.

Carroll, J. M., & Rosson, M. B. (2007). Participatory design in community informatics. *Design studies*, *28*(3), 243–261.

Cheng, S., Wang, J., Shen, X., Chen, Y., & Dey, A. (2022, 06). Collaborative eye tracking based code review through real-time shared gaze visualization. *Frontiers of Computer Science*, *16*. doi: 10.1007/s11704-020-0422-1

Clark, B., & Sharif, B. (2017). itracevis: Visualizing eye movement data within eclipse. In *2017 ieee working conference on software visualization (vissoft)* (p. 22-32). doi: 10.1109/VISSOFT.2017.30

Clement, A. (1996). Computing at work: empowering action by "low-level users". *Computerization and controversy: value conflicts and social choices*, 383.

Ehn, P., & Sandberg, Å. (1979). *Företagsstyrning och löntagarmakt: planering, datorer, organisation och fackligt utredningsarbete.* Prisma i samarbete med Arbetslivscentrum.

Figma. (n.d.). *Figma: The collaborative interface design tool.* Retrieved from `https://www.figma.com/`

Glücker, H., Raab, F., Echtler, F., & Wolff, C. (2014). Eyede: gaze-enhanced software development environments. In *Chi'14 extended abstracts on human factors in computing systems* (pp. 1555–1560).

Green, C., Jaspan, C., Hodges, M., He, L., Shen, D., & Zhang, N. (2023). Developer productivity for humans, part 5: Onboarding and ramp-up. *IEEE Software*, *40*(5), 13-19. doi: 10.1109/MS.2023.3291158

Guilizzoni, P. (n.d.). *What are Wireframes?* . Retrieved from `https://balsamiq.com/learn/articles/what-are-wireframes/`

Gumm, D. C., Janneck, M., & Finck, M. (2006). Distributed participatory design–a case study. In *Proceedings of the dpd workshop at nordichi* (Vol. 2).

Hellman, J., Cheng, J., & Guo, J. L. (2021). Facilitating asynchronous participatory design of open source software: Bringing end users into the loop. In *Extended abstracts of the 2021 chi conference on human factors in computing systems* (pp. 1–7).

Hijazi, H., Cruz, J., Castelhano, J., Couceiro, R., Castelo-Branco, M., de Carvalho, P., & Madeira, H. (2021). ireview: an intelligent code review evaluation tool using biofeedback. In *2021 ieee 32nd international symposium on software reliability engineering (issre)* (p. 476-485). doi: 10.1109/ISSRE52982.2021.00056

Hull, M. F. (2021). The role of technical communicators in open-source software: A systematic review.

Iivari, N. (2009). "constructing the users" in open source software development: An interpretive case study of user participation. *Information Technology & People*, *22*(2), 132–156.

Johnson, B., Song, Y., Murphy-Hill, E., & Bowdidge, R. (2013). Why don't software developers use static analysis tools to find bugs? In *2013 35th international conference on software engineering (icse)* (p. 672-681). doi: 10.1109/ICSE.2013.6606613

Johnson, J., & Henderson, A. (2002, jan). Conceptual models: begin by designing what to design. *Interactions*, *9*(1), 25–32. Retrieved from `https://doi.org/10.1145/503355.503366` doi: 10.1145/503355.503366

Kuang, P., Söderberg, E., & Höst, M. (2024). Developers' perspective on today's and tomorrow's programming tool assistance: A survey. In *10th edition of the programming experience workshop, px/24.*

Kuang, P., Söderberg, E., Niehorster, D. C., & Höst, M. (2023). Toward gaze-assisted developer tools. In *2023 ieee/acm 45th international conference on software engineering: New ideas and emerging results (icse-nier)* (p. 49-54). doi: 10.1109/ICSE-NIER58687.2023.00015

Letondal, C., & Mackay, W. E. (2004). Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment. In (p. 31–41). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1011870.1011875` doi: 10.1145/1011870.1011875

Liang, J. T., Yang, C., & Myers, B. A. (2024, apr). 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)* (p. 605-617). Los Alamitos, CA, USA: IEEE Computer Society. Retrieved from `https://doi.ieeecomputersociety.org/`

Maudet, N., Leiva, G., Beaudouin-Lafon, M., & Mackay, W. (2017). Design breakdowns: Designer-developer gaps in representing and interpreting interactive systems. In *Proceedings of the 2017 acm conference on computer supported cooperative work and social computing* (p. 630–641). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2998181.2998190` doi: 10.1145/2998181.2998190

McCabe, A. T., Söderberg, E., Church, L., & Kuang, P. (2022). Visual cues in compiler conversations. In S. Holland, M. Petre, L. Church, & M. Marasoiu (Eds.), *Proceedings of the 33rd annual workshop of the psychology of programming interest group, PPIG 2022, the open university, milton keynes, UK & online, september 5-9, 2022* (pp. 25–38). Psychology of Programming Interest Group. Retrieved from `https://ppig.org/papers/2022-ppig-33rd-mccabe/`

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *11*(3), 309–346.

Myers, B. A., Ko, A. J., LaToza, T. D., & Yoon, Y. (2016). Programmers are users too: Human-centered methods for improving programming tools. *Computer*, *49*(7), 44-52. doi: 10.1109/MC.2016.200

Oviatt, S., & Cohen, P. (2000). Perceptual user interfaces: multimodal interfaces that process what comes naturally. *Communications of the ACM*, *43*(3), 45–53.

Paudyal, B., Creed, C., Frutos-Pascual, M., & Williams, I. (2020). Voiceye: A multimodal inclusive development environment. In *Proceedings of the 2020 acm designing interactive systems conference* (pp. 21–33).

Pruitt, J., & Grudin, J. (2003). Personas: practice and theory. In (p. 1–15). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/997078.997089` doi: 10.1145/997078.997089

Radevski, S., Hata, H., & Matsumoto, K. (2016). Eyenav: Gaze-based code navigation. In *Proceedings of the 9th nordic conference on human-computer interaction.* New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/2971485.2996724` doi: 10.1145/2971485.2996724

Rogers, Y., Sharp, H., & Preece, J. (2011). *Interaction Design: Beyond Human-Computer Interaction.* Retrieved from `http://discovery.ucl.ac.uk/1326236/`

Santos, A. L. (2021). 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* (p. 31–35). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3464432.3464435` doi: 10.1145/3464432.3464435

Saranpää, W., Apell Skjutar, F., Heander, J., Söderberg, E., Niehorster, D. C., Mattsson, O., ... Church, L. (2023). Gander: a platform for exploration of gaze-driven assistance in code review. In *Proceedings of the 2023 symposium on eye tracking research and applications.* New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3588015.3589191` doi: 10.1145/3588015.3589191

Schuler, D., & Namioka, A. (1993). *Participatory design.*

Shakil, A., Lutteroth, C., & Weber, G. (2019). Codegazer: Making code navigation easy and natural with gaze input. In *Proceedings of the 2019 chi conference on human factors in computing systems* (p. 1–12). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/3290605.3300306` doi: 10.1145/3290605.3300306

Siau, K., & Wang, W. (2020). Artificial intelligence (ai) ethics: ethics of ai and ethical ai. *Journal of Database Management (JDM)*, *31*(2), 74–87.

Spinuzzi, C. (2005). The methodology of participatory design. *Technical communication*, *52*(2), 163–174.

Talon. (2024). *Talon: Powerful Hands-free Input.* (Retrieved 3 Apr, 2024 from `https://talonvoice.com/`)

Tobii. (2023). *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/` (Feb 15, 2023))

Wikipedia. (2024, 5). *Participatory design.* Retrieved from `https://en.wikipedia.org/wiki/Participatory_design`

Wubishet, Z. S., Bygstad, B., & Tsiavos, P. (2013). A participation paradox: Seeking the missing link between free/open source software and participatory design. *Journal of Advances in Information Technology*, *4*(4), 181–193.

Yang, J., Jimenez, C. E., Wettig, A., Yao, S., Narasimhan, K., & Press, O. (2024). *Swe-agent: Agent computer interfaces enable software engineering language models.*

Zhang, Y., Ruan, H., Fan, Z., & Roychoudhury, A. (2024). *Autocoderover: Autonomous program improvement.*