Towards Personalized and Contextualized Code Explanations

Michelle Brachman
IBM Research
Cambridge, MA, USA
michelle.brachman@ibm.com

Arielle Goldberg
IBM
Poughkeepsie, NY, USA
Arielle.Goldberg1@ibm.com

Andrew Anderson
IBM Research
San Jose, CA, USA
Andrew.Anderson2@ibm.com

Stephanie Houde
IBM Research
Cambridge, MA, USA
Stephanie.Houde@ibm.com

Michael Muller
IBM Research
Cambridge, MA, USA
michael_muller@us.ibm.com

Justin D. Weisz
IBM Research
Yorktown Heights, NY, USA
jweisz@us.ibm.com

Abstract

Code understanding is a common and important use case for generative AI code assistance tools. Yet, a user's background, context, and goals may impact the kinds of code explanations that best fit their needs. Our aim was to understand the kinds of configurations users might want for their code explanations and how those relate to their context. We ran an exploratory study with a medium-fidelity prototype and 10 programmers. Participants valued having configurations and desired automated personalization of code explanations. They found particular merit in being able to configure the structure and detail level in code explanations and felt that their needs might change depending on their prior experience and goals.

CCS Concepts

• **Human-centered computing → Empirical studies in HCI**.

Keywords

Code Explanations, Personalization, Generative AI

ACM Reference Format:
Michelle Brachman, Arielle Goldberg, Andrew Anderson, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2025. Towards Personalized and Contextualized Code Explanations. In *Adjunct Proceedings of the 33rd ACM Conference on User Modeling, Adaptation and Personalization (UMAP Adjunct '25), June 16–19, 2025, New York City, NY, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3708319.3733681

1 Introduction

Generative AI has made code assistants accessible and useful to professionals who work with code. Tools, such as Github CoPilot[1] and IBM Watsonx Code Assistant[2], typically provide a variety of functionalities, such as code generation, test generation, and code explanations. Generating code explanations is a popular and valuable function of these tools [2, 26], due to the challenges of understanding unfamiliar code [17]. While code assistants provide basic functionality for supporting code understanding, including more context [2, 4, 7, 17, 21, 24], as well as supporting adaptation and control over explanations [5, 27] could be beneficial.

Code understanding is a complex process and a user's needs can be impacted by a variety of factors related to: users' backgrounds, the code, task, and project. Personalization and customization may be possible using Generative AI [1, 11, 15], such as by adapting code explanations to individual user needs using Large Language Models (LLMs) [3] or allowing users to provide natural language instructions about their needs and preferences[3]. We hypothesized that if we knew more about what factors impact users' explanation customization needs, future systems could better automate the personalization and contextualization of code explanations or assistance.

Our research questions were: 1) how valuable is having code explanation options and which options are most valuable, 2) what aspects of users or their tasks impact preferences around explanation options, and 3) what kinds of automated adaptation and control do users prefer? To answer these questions, we ran a formative think-aloud study with a medium-fidelity prototype of a code assistant that supports configuration of code explanation options. Participants could change the structure, detail level, and style of the code explanations, as well as add on language concept descriptions, examples, or real-world analogies. We asked participants about their preferences and potential use for this kind of tool.

Users valued having options for code explanations, in particular for the structure and detail level. They felt that the background of a user, the user's goals, and the project context impact the kinds of personalization needed. Automation will likely be valuable, as long as the system provides the ability to control and modify the explanations. In this paper, we contribute the findings and analysis of this exploratory study, which provide insight into how code assistance tools can support the individual needs of users.

2 Related Work

Our work builds upon and extends recent research around generative AI code tools and personalization for AI code tools.

Generative AI code assistance tools typically provide the capability to get explanations of code, such as through conversation [2, 18, 20] or by clicking a button [24, 28]. Several studies have validated the potential of LLM-generated code explanations [9] and

[1]https://github.com/features/copilot
[2]https://www.ibm.com/products/watsonx-code-assistant

[3]https://www.cursor.com

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
UMAP Adjunct '25, New York City, NY, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1399-6/25/06
https://doi.org/10.1145/3708319.3733681

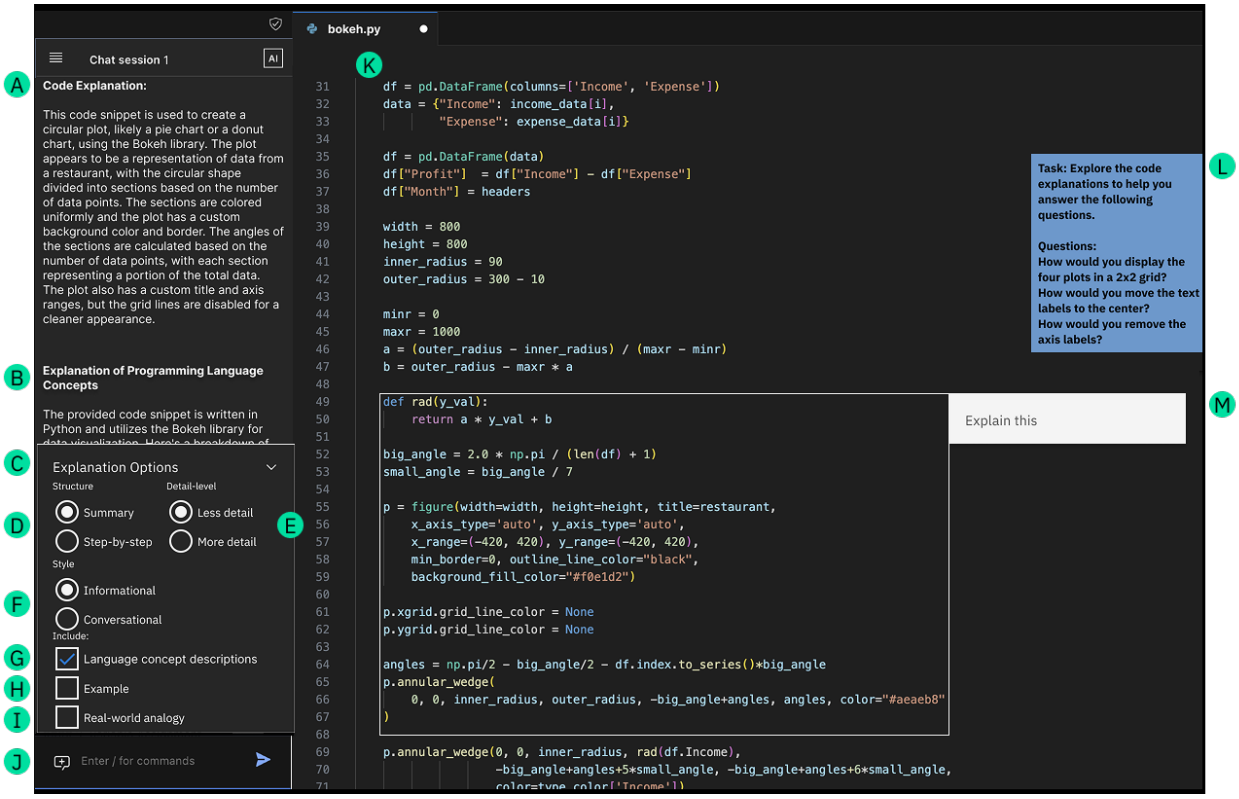


Figure 1: Figma prototype used for the study. The code explanations are shown in the left panel and the code is on the right.

code explanations with context [7]. One tool for code understanding enables users to request more information, such as about the concept or usage, with the click of a button, which increased task completion but not understanding [17]. Another system provides helpful explanations alongside generated code, but could also benefit from expandable, adaptable, and controllable explanations [27]. Finally, several studies have indicated that users prefer concise code explanations [2, 27]. Our work contributes to the understanding of users' needs for generated code explanations.

Recent work emphasizes a need for personalization of generative AI assistance for code. Personalization of the user's experience may follow a strategy of user modeling as part of Theory of Mind [25], in which the system learns the user's characteristics and adapts its display and behavior to the user's needs. Studies of new code assistance tools have demonstrated a need for supporting differing experience levels [4] and tasks [18], which can be described as relatively invariant *traits* and rapidly changing *states*, respectively (e.g., [6]). Studies also propose that future work should leverage code context, project context, system context, and users' personal context to improve code assistance tools [2, 5, 17, 21]. Several tools and studies have begun to integrate adaptations of code explanations. One code understanding tool used various aspects of a user's mental state, such as their knowledge and experience, to personalize code explanations, though it didn't impact understanding [19]. Other research has found potential value in adapting to the user's intent [18], problem-solving style [3], task description [28], or the user's background [16]. Our work contributes by investigating users' preferences for personalizing code explanations and the kinds of context that might impact their needs.

3 Prototype

We describe our prototype's design rationale and provide a detailed description of the prototype's features (see Figure 1).

3.1 Design rationale

We designed our code explanation tool to be a variant on existing systems, with code explanations appearing in a side panel. We based our explanation options on prior work around the kinds of code explanations generated by LLMs [3, 12–14] and needs for code explanations [8, 22]. We extracted the qualities, content, and styles of code explanations described and grouped them. We distilled the ideas into three binary options users could choose for the main explanation: structure (summary/step-by-step), detail level (less/more), and style (informational/conversational). We also provided three "add-ons": programming concept descriptions (aimed at providing more information for those with less experience or knowledge in the language), example, and real-world analogy.

3.2 Description

We created a Figma[4] prototype (see Figure 1) designed to simulate an extension to the Visual Studio Code IDE[5]. Our design was primarily motivated by the need to be easy to understand in order to elicit feedback focused on the value of the explanation options themselves and the ability to modify explanations, rather than the visual design.

The code explanation extension is the left panel (A-J) and the code is on the right (K). The main code explanations always show up at the top of the left panel (A) based on the selected options from *structure* (D), *detail level* (E), and *style* (F). For *structure*, users can choose *summary* (default) or *step-by-step* (D). For *detail level*, they can choose *less detail* (default) or *more detail* (E). For *style*, they can choose *informational* (default) or *conversational* (F). Participants could also select to include *Language concept descriptions* (G), an *Example* (H), or a *Real-world analogy* (I). These appeared below the main explanation (B). The add-ons were adapted for the different code snippets but did not have the *structure*, *detail-level*, or *style* adaptations for simplicity. There is also a static input box (J) to indicate that a future version of this kind of tool would support text input. The prototype has a task description (L) and a button to click to view the explanation for a part of the code (M).

We pre-generated the explanations using Llama-3.1-70b using the prompt: "Explain the following code:[source code from [17]]. Additionally, follow the following instructions: [based on configuration] Only provide the explanation- do not provide a simplified version of the code." The instructions changed depending on the options configured. For example, for summary, informational, and low-detail, the instructions were: "Provide the explanation as a summary instead of step-by-step. Provide the explanation in an informational style. Provide a concise explanation." Table 1 shows examples of generated explanations for different configurations. For language concept descriptions, the instructions were "For the following code, provide an explanation of the programming language concepts in the code." For examples, the instructions were "For the following code, provide an example of inputs and outputs to the code." For real-world analogies, the instructions were " For the following code, provide a real world analogy of the code."

4 Methods

We ran a formative think-aloud study using the prototype described above (section 3). Participants stepped through a code task using the prototype and told us about their experience and preferences.

4.1 Study Protocol

The study was a supervised interview via a recorded video call. We began with general profile questions to understand the participant's background and prior experience with generative AI tools. Next, we demonstrated how to use the prototype using a demo task (from [17]). After the participant was comfortable using the tool, they worked on the main task for up to 25 minutes. Participants had a 101-line Python code file that used the Bokeh library and a task to answer three questions (see Figure 1-L, adapted from [17]). Participants could view the explanations for the whole code file or three smaller snippets of code (pre-selected by the researchers) and change the explanation options. We prompted the participants to explore at least two code snippets if they hadn't done so and to explain their reasoning and preferences around their use of the explanation options (see Figure 1-C). Finally, we asked participants follow-up questions about their experience, including why they chose the options they did, how valuable they felt the ability to customize code explanations was, and their preferences around automated personalization of code explanations.

4.2 Participants

We recruited participants from IBM who program at work through an internal channel. Participants were from countries in the Asian Pacific (7) and the Americas (3). They ranged in experience at their job (<6 months: 1, 6 months - 1 year: 1, 1-2 years: 2, 3-5 years: 4, 6-10 years: 1, 16-20 years: 1). Participants also ranged in experience with the Python language (never: 1, used once or twice: 2, experience but not daily: 4, daily use: 3). None had used the Python library Bokeh. Participants used generative AI tools for code assistance daily (3), sometimes but not daily (4), in the past (1), once or twice (1), and never (1).

4.3 Data and Analysis

We collected auto-transcriptions and repaired them manually as needed. We then used affinity diagramming to perform an inductive qualitative analysis [23]. Three researchers worked together on the analysis. We first divided and individually grouped and coded the 10 transcripts, before working together to merge groups, iterate and refine codes and themes, and discuss the coding of transcript segments until there was agreement.

5 Results

5.1 Value and Preferences

We asked participants about the value of being able to customize code explanations through options as well as which options were most and least important to have in such a system[6].

5.1.1 Having explanation options is valuable. Participants generally found having ways to adapt and modify the explanations to be valuable. On a scale from 0 (not valuable at all) to 10 (most valuable), participants all rated having options for their explanations between 7 and 10 ($mean = 9.30, SD = 1.06$). They valued having options and the ability to change what they were looking at, as well as to customize the explanations to their particular needs. P5 said: "Maybe by customizing it like from my own specific needs, I can get the exact thing which I was looking for." Several participants also talked about valuing the ability to select the options without needing to write prompts to change the explanations.

5.1.2 Structure and detail level most valuable. Participants had the most positive comments about the *structure* and *detail level* options. In particular, many preferred the *step-by-step* structure and *more detail* choices. All 10 participants had positive comments about the value of the *step-by-step structure* option. Three participants also

[4]https://www.figma.com/
[5]https://code.visualstudio.com

[6]We did not ask each participant about every option, but allowed them to talk about what was most and least important to them.

Table 1: Examples of explanations generated for a snippet of code and four sets of configurations. Note, the step-by-step explanations are truncated and only show step 1 for brevity.

Configuration	Explanation
Summary, less detail, informational	This code snippet is used to position and orient text labels around a circular plot, likely a pie chart or a polar plot. It calculates the x and y coordinates of the labels based on the last radius value and the angles of the labels, taking into account a large angle offset (*big_angle*). The labels are then positioned at these coordinates with their text aligned to the left and baseline centered, and rotated to match the angle of the corresponding section of the plot. The rotation is adjusted to ensure that labels are always oriented in a readable direction, with those below the horizontal axis rotated to point upwards.
Summary, less detail, conversational	This code is essentially positioning and rotating text labels around a circle. It's calculating the x and y coordinates of the labels based on a radius and a set of angles, then adjusting those angles to ensure the labels are always readable (i.e., not upside down). The labels are then placed at those coordinates, rotated to match the angle of the circle at that point, and formatted with a specific font size and alignment.

Step-by-step, less detail, informational	**Code Explanation: Plotting Text Labels on a Circular Plot**
	This code snippet is used to plot text labels on a circular plot, likely a pie chart or a polar plot. Here's a step-by-step breakdown of what the code does:
	Step 1: Calculate x and y Coordinates for Text Labels
	`xr = radii[-1]*np.cos(np.array(-big_angle/2 + angles))`
	`yr = radii[-1]*np.sin(np.array(-big_angle/2 + angles))`
	In this step, the code calculates the x and y coordinates (xr and yr) for the text labels. The coordinates are calculated using the parametric equation of a circle, where $radii[-1]$ is the radius of the circle, and *angles* is an array of angles in radians. The $-big_angle/2$ term is likely used to offset the angles to start from a specific position on the circle. ...

Step-by-step, more detail, informational	**Code Explanation: Plotting Text Labels on a Circular Plot**
	The provided code snippet is used to plot text labels on a circular plot, likely a pie chart or a polar plot. The labels are positioned at specific angles and radii, with their orientation adjusted to ensure readability.
	Step 1: Calculating x and y Coordinates for Label Positions
	`xr = radii[-1]*np.cos(np.array(-big_angle/2 + angles))`
	`yr = radii[-1]*np.sin(np.array(-big_angle/2 + angles))`
	In this step, the code calculates the x and y coordinates for the positions of the text labels. The $radii[-1]$ variable represents the radius of the circle where the labels will be placed. The *np.cos* and *np.sin* functions are used to calculate the x and y coordinates, respectively, based on the angles.
	The angles are calculated by adding the *angles* array to $-big_angle/2$. This suggests that the labels are being positioned at specific angles, offset by half of the *big_angle*. The *np.array* function is used to ensure that the angles are treated as an array, allowing for element-wise operations. ...

talked about the *summary structure* being helpful, especially as a starting point. Three participants didn't feel that having a *structure* option was critical, because they could get a *summary* from reading through the *step-by-step* or *less detail* explanations. Similarly, most participants talked about the *more detail* detail level as valuable (9). While five participants talked about *less detail* being helpful, four participants talked about not needing the *less detail* option because "it's better always to have more details" (P1).

5.1.3 Language concept descriptions most useful add-on. Of the three explanation add-ons (*language concept descriptions*, *examples*, and *real-world analogy*), participants were most positive about *language concept descriptions* (4 positive, 2 negative) compared to example (1 positive, 1 negative) or real-world analogy (2 negative). P4 said that the *language concept description* is "providing you the details of each of the line [of code]." Two participants also said that they would want all three of the add-ons selected.

5.1.4 Style not critical. Participants more often preferred an *informational* style (4), but some also found value in the *conversational* style (2). Yet, even those with preferences often stated that they could be okay with either *style* and six participants talked about not needing *style* as an option.

5.2 Context impacts needs

All participants talked about the kinds of context that might impact their needs for explanation options: their current goal (10), the background of the user (8), and the project context (7).

5.2.1 Current goal. Participants talked about their goals impacting their explanation needs. Participants often talked about getting the high-level idea or the gist of the code (5). Once they have the high-level idea, many want to be able to move to a more focused or detailed understanding of the code (8). Getting more details might involve understanding a particular concept or piece of code (9),

understanding the output of the code (6), connecting specific code snippets to explanations (5), looking for functionality (4), or trying to get a complete grasp of the code (2). For example, P6 selected options as a starting point and to find the right part of the code: "I always started with summary and less details to just see if what I'm looking at is even the part where I should be looking at."

5.2.2 User background/knowledge. Different kinds of developer experiences impact the kinds of explanations users might need. Knowledge of particular programming languages or libraries (5), general programming experience (4), and knowledge of the specific code (4) all might impact users' explanation needs.

5.2.3 Project context. Participants talked about qualities of the code impacting their explanation needs, like the size of the code, complexity, or readability (5). Three participants commented that how rushed they were might change the kinds of explanations they needed.

5.3 Interactions with the system

Towards designing future systems with explanation customization, we asked participants about their preferences around automation of the customization and how often they imagined changing options.

5.3.1 Desire for automation. Four participants were positive about automating the personalization, though two of those talked about certain situations or options that might lend themselves more toward automation. Three more participants talked about automation under certain circumstances and two participants talked about wanting to start by manually selecting options and having the system learn. Six participants talked about wanting to be able to intervene in an automated personalization of code explanations. Two participants wanted to control the options themselves.

5.3.2 Frequency of option changes. Participants often talked about wanting to set options as part of a profile (6), but some thought the options would be useful on a case-by-case basis. For example, P2 said "it would be nice to be able to toggle between a summary and a step-by-step when either of those apply."

6 Discussion

6.1 Design Implications

Participants generally found the ability to change the structure (summary/step-by-step) and detail level (less/more) as most valuable. Relatedly, programming concept descriptions, which essentially provided more and different kinds of details, were the most useful add-on. While some participants' preferences aligned with prior work showing desires for concise explanations [2, 27], participants typically preferred having *more detail*. One reason for this difference may be participants' lack of familiarity with the code. Design of systems should, at a minimum, allow users to change the way code explanations are presented, the level of detail, and the kinds of detail, based on their background and tasks. Further, our study highlights and provides direct evidence supporting prior suggestions of the importance of individual and contextual differences in code explanation needs [2, 4, 5, 17, 18, 21], as not all participants had the same needs and those needs will likely change over time.

Participants felt that some level of automated personalization would be useful to support their changing needs, where option selections would likely adapt but be consistent for each user. In particular, the user's current goal was most often discussed as critical in which kinds of explanations users needed, with different kinds of goals impacting the structure, detail level, and kinds of details users needed. This aligns and builds upon work using Mutual Theory of Mind for code assistant personalization [16, 19], with the user's coding-familiarity as a relatively static trait-based aspect of the user model, and the user's current task as a relatively dynamic state-based aspect of the user model [25]. At the same time, users wanted to maintain the ability to intervene to change automated settings when their task required something different. Design of systems for this purpose should combine automated adaptations with the ability for users to directly modify those options.

6.2 Limitations

In this study, participants used a medium-fidelity prototype for one small code task in which they were unfamiliar with the code and Python library. Participants weren't able to make changes or execute the code, limiting our findings. While the importance of users' goals will likely remain, different contexts, such as larger code bases, will likely lead to different specific goals and needs [8, 10].

7 Conclusion

Recent research has suggested that code assistance tools will likely benefit from personalization that leverages user and task context. Our goal was to understand the kinds of personalization programmers want for code explanations. Participants valued being able to select options for their explanations and, in particular, wanted to be able to change the structure and detail level of the explanations. Participants also felt that their goals in the task, own background, and project context were important factors that impact their code explanation needs. Participants were positive about the possibility of automatic personalization of code explanations, as long as they can make changes. These findings support the need for personalization and context in code assistance and provide directions for future tools to better support the individual needs of programmers.

References

[1] Mathieu Acher. 2024. A demonstration of end-user code customization using generative ai. In *Proceedings of the 18th International Working Conference on Variability Modelling of Software-Intensive Systems*. 139–145.
[2] Mehmet Akhoroz and Caglar Yildirim. 2025. Conversational AI as a Coding Assistant: Understanding Programmers' Interactions with and Expectations from Large Language Models for Coding. *arXiv preprint arXiv:2503.16508* (2025).
[3] Andrew Anderson, David Piorkowski, Margaret Burnett, and Justin Weisz. 2025. An LLM's Attempts to Adapt to Diverse Software Engineers' Problem-Solving Styles: More Inclusive & Equitable? *arXiv preprint arXiv:2503.11018* (2025).
[4] Yasharth Bajpai, Bhavya Chopra, Param Biyani, Cagri Aslan, Dustin Coleman, Sumit Gulwani, Chris Parnin, Arjun Radhakrishna, and Gustavo Soares. 2024. Let's Fix this Together: Conversational Debugging with GitHub Copilot. In *2024 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–12.
[5] Shraddha Barke, Michael B James, and Nadia Polikarpova. 2023. Grounded copilot: How programmers interact with code-generating models. *Proceedings of the ACM on Programming Languages* 7, OOPSLA1 (2023), 85–111.
[6] Michael Braun and Florian Alt. 2019. Affective assistants: A matter of states and traits. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–6.
[7] Eason Chen, Ray Huang, Han-Shin Chen, Yuen-Hsien Tseng, and Liang-Yi Li. 2023. GPTutor: a ChatGPT-powered programming tool for code explanation. In

International Conference on Artificial Intelligence in Education. Springer, 321–327.

[8] Amy J Ko, Robert DeLine, and Gina Venolia. 2007. Information needs in collocated software development teams. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 344–353.

[9] Knud Ronau Larsen and Magnus Edvall. 2024. Investigating the impact of Generative AI on newcomers' understanding of Software Projects.

[10] Thomas D LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. 492–501.

[11] Gun Ho Lee, Kyoung Jun Lee, Baek Jeong, and Taekyung Kim. 2024. Developing personalized marketing service using generative AI. *IEEE Access* 12 (2024), 22394–22402.

[12] Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew Tran, and Arto Hellas. 2023. Comparing code explanations created by students and large language models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*. 124–130.

[13] Stephen MacNeil, Andrew Tran, Arto Hellas, Joanne Kim, Sami Sarsa, Paul Denny, Seth Bernstein, and Juho Leinonen. 2023. Experiences from using code explanations generated by large language models in a web software development e-book. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 931–937.

[14] Stephen MacNeil, Andrew Tran, Dan Mogil, Seth Bernstein, Erin Ross, and Ziheng Huang. 2022. Generating diverse code explanations using the gpt-3 large language model. In *Proceedings of the 2022 ACM conference on international computing education research-volume 2*. 37–39.

[15] Sandra C Matz, Jacob D Teeny, Sumer S Vaid, Heinrich Peters, Gabriella M Harari, and Moran Cerf. 2024. The potential of generative AI for personalized persuasion at scale. *Scientific Reports* 14, 1 (2024), 4692.

[16] Dario Andres Silva Moran, Kristina Brimijoin, Gabriel Enrique Gonzalez, Stephanie Houde, Michael Muller, Michelle Brachman, and Justin D Weisz. 2025. ACE: Moving toward Co-Investigation with the Agentic Code Explorer. (2025).

[17] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[18] Kevin Pu, Daniel Lazaro, Ian Arawjo, Haijun Xia, Ziang Xiao, Tovi Grossman, and Yan Chen. 2025. Assistance or Disruption? Exploring and Evaluating the Design and Trade-offs of Proactive AI Programming Support. *arXiv preprint arXiv:2502.18658* (2025).

[19] Jonan Richards and Mairieli Wessel. 2024. What you need is what you get: Theory of mind for an llm-based code understanding assistant. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 666–671.

[20] Steven I Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D Weisz. 2023. The programmer's assistant: Conversational interaction with a large language model for software development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces*. 491–514.

[21] Agnia Sergeyuk, Yaroslav Golubev, Timofey Bryksin, and Iftekhar Ahmed. 2025. Using AI-based coding assistants in practice: State of affairs, perceptions, and ways forward. *Information and Software Technology* 178 (2025), 107610.

[22] Jonathan Sillito, Gail C Murphy, and Kris De Volder. 2006. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. 23–34.

[23] David R Thomas. 2006. A general inductive approach for analyzing qualitative evaluation data. *American journal of evaluation* 27, 2 (2006), 237–246.

[24] Christoph Treude and Marco A Gerosa. 2025. How Developers Interact with AI: A Taxonomy of Human-AI Collaboration in Software Engineering. *arXiv preprint arXiv:2501.08774* (2025).

[25] Qiaosi Wang, Sarah Walsh, Mei Si, Jeffrey Kephart, Justin D Weisz, and Ashok K Goel. 2024. Theory of mind in human-ai interaction. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–6.

[26] Justin D Weisz, Shraddha Kumar, Michael Muller, Karen-Ellen Browne, Arielle Goldberg, Ellice Heintze, and Shagun Bajpai. 2024. Examining the Use and Impact of an AI Code Assistant on Developer Productivity and Experience in the Enterprise. *arXiv preprint arXiv:2412.06603* (2024).

[27] Litao Yan, Alyssa Hwang, Zhiyuan Wu, and Andrew Head. 2024. Ivie: Lightweight anchored explanations of just-generated code. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 1–15.

[28] Sebastian Zhao, Alan Zhu, Hussein Mozannar, David Sontag, Ameet Talwalkar, and Valerie Chen. 2025. CodingGenie: A Proactive LLM-Powered Programming Assistant. *arXiv preprint arXiv:2503.14724* (2025).