Designing PairBuddy – A Conversational Agent for Pair Programming

PETER ROBE, University of Tulsa
SANDEEP KAUR KUTTAL, University of Tulsa

From automated customer support to virtual assistants, conversational agents have transformed everyday interactions, yet despite phenomenal progress, no agent exists for programming tasks. To understand the design space of such an agent, we prototype PairBuddy – an interactive pair programming partner – based on research from conversational agents, software engineering, education, human-robot interactions, psychology, and artificial intelligence. We iterated PairBuddy's design using a series of Wizard-of-Oz studies. Our pilot study of six programmers showed promising results and provided insights toward PairBuddy's interface design. Our second study of fourteen programmers was positively praised across all skill levels. PairBuddy's active application of soft skills – adaptability, motivation, and social presence – as a navigator increased participants' confidence and trust, while its technical skills – code contributions, just-in-time feedback, and creativity support – as a driver helped participants realize their own solutions. PairBuddy takes the first step towards an Alexa-like programming partner.

CCS Concepts: • **Human-centered computing** → **User studies**.

Additional Key Words and Phrases: Conversational agents, pair programming, user centered design, Wizard of Oz.

ACM Reference Format:
Peter Robe and Sandeep Kaur Kuttal. 2021. Designing PairBuddy – A Conversational Agent for Pair Programming. 1, 1 (September 2021), 36 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Conversational agents allow humans to use natural language to directly interface with computer agents such as virtual assistants (e.g., Apple's Siri [189], Google Assistant [190], and Amazon's Alexa [188]), customer support agents, or individual/social chatbots (e.g., Mitsuku [167], Cleverbot [166], and XiaoIce [169]). Conversational agents mimic human conversations, establish more personal connections, and can even increase accessibility for people with physical disabilities or language barriers. For businesses, conversational agents personalize customer experience, while bringing down operational costs. Today, a full two-thirds of the most popular websites use conversational agents to interact with users and address their needs [165]. Despite the phenomenal penetration of conversational agents in domains ranging from business to personal use and entertainment, no conversational agents exist for computer programming tasks.

In this paper, we take the first step towards the creation of "PairBuddy" – a conversational agent that enables interactive communication between a human programmer and a computer. We

Authors' addresses: Peter Robe, pjr144@utulsa.edu, University of Tulsa, Tulsa, Oklahoma, 74104; Sandeep Kaur Kuttal, sandeep-kuttal@utulsa.edu, University of Tulsa, Tulsa, Oklahoma, 74104.

XXXX-XXXX/2021/9-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

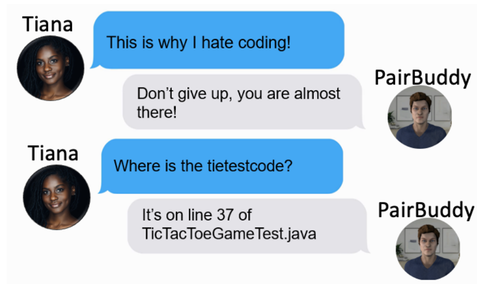

Fig. 1. Tiana interacting with PairBuddy.

designed PairBuddy to promote effective programming by using "pair programming," an established collaboration technique used in education and the industry.

In pair programming, two programmers work collaboratively on the same design, algorithm, code, or test [143, 200, 201]. Programmers switch between the roles of driver (writing code) and navigator (making suggestions). Pair programming provides a variety of benefits, including increased code quality, productivity, creativity, knowledge management, and self-efficacy [20, 33, 40, 47, 51, 98, 124, 125, 143, 152, 156, 197, 200–202, 216]. It even has the potential to reduce gender prejudice by encouraging women to pursue computer science [197]. Pair programming increases programmers' contemporary skills, understanding of fundamental concepts, and intellectual pursuits [33, 124, 152, 156, 197]. However, pair programming has certain limitations including scheduling difficulties, collocating pairs, student resistance to pairing, and the dependency on a partner's programming abilities [68, 82, 139, 199]. We conjecture that PairBuddy's design will promote the benefits of pair programming while reducing its limitations.

1.1 Motivational Scenario

To motivate the need and inform the abilities of PairBuddy, we provide the following scenario:

Tiana is a junior CS student who enjoys pair programming, but due COVID-19, she has to return home. Given the physical and time-zone differences, scheduling pair programming sessions is hard. Additionally, all of her potential partners either like to work solo, have incompatible cognitive styles, criticize without providing solutions (e.g., *"We definitely cannot finish this"*), are overly competitive, take credit for Tiana's work, or have other problematic habits. As a result, their sessions with Tiana are stressful and unproductive, resulting in the loss of pair programming's benefits. Fortunately, Tiana can use PairBuddy (Figure 1).

Tiana can collaborate with PairBuddy, switching between the roles of driver and navigator to create test cases, write code, and refactor code. Additionally, PairBuddy: 1) is unbiased toward Tiana's gender, ethnicity, and socio-economic status; 2) motivates and encourages her hard work; 3) gives non-judgemental feedback, allowing Tiana to ignore or disagree without the risk of hurt feelings; 4) provides a sense of security through its active listening and engagement through both voice and text; 5) encourages and instills healthy problem-solving and creative styles; and 6) avoids dominating power dynamics by balancing pair programming roles. For these reasons and more, Tiana prefers PairBuddy as her programming partner to complete difficult assignments comfortably and effectively.

1.2 Challenges

Since no pair programming conversational agents exist, the challenges of creating such an agent include:

(1) **An Unknown Design – Interface and Interactions:** PairBuddy's interface and interaction design must be explored due to the unique properties of pair programming dialogue [151]. To support programmer-computer interaction, PairBuddy must imitate an effective programmer by integrating a multitude of technical and soft skills, including diverse problem-solving and creative strategies. Therefore, PairBuddy's design must be informed from multi-disciplinary research.
(2) **A Specific Domain – Software Development:** PairBuddy needs to be created for the specific domain of software development, which remains relatively unexplored by conversational agent research. Developing software is unique, as it demands the synthesis of requirements; the generation and development of solutions; and the implementation, testing and refactoring of code. Therefore, pre-existing conversational agents cannot be directly modified for programming, and instead, a new paradigm must emerge to support programmer collaboration with an agent. This shift in approach must be informed from software engineering and education. Specifically, we consider literature on Intelligent Tutoring Systems to integrate the programming and educational aspects of pair programming.
(3) **A Specific Userbase – Programmers:**
Programmers are a unique population that have yet to be studied in the realm of conversational agents. From students to professionals, programmers' experiences are diverse; some may use agents to learn programming concepts, while others may develop those same agents. To design PairBuddy for both populations, we must conduct user studies to fully understand the breadth of expectations, preferences, and needs of programmers.

These challenges necessitate a review of existing research on human-computer interactions, human-robotic interactions, software engineering, conversational agents, artificial intelligence, intelligent tutoring systems, education, psychology, cognitive science, and management science to inform the design of PairBuddy.

2 WIZARD OF OZ STUDIES

Due to the limited available research on pair programming conversational agents, we used an iterative approach to understand and design PairBuddy through a series of two Wizard of Oz studies with programmers.

In a Wizard of Oz study, participants interact with an agent whose actions are secretly controlled by a human "wizard." Wizard of Oz is a rapid-prototyping method that examines interfaces that are technically demanding or are yet to be created [76]. It helps develop user-friendly interfaces that promote natural language dialogue, consider the unique qualities of human-agent interaction as distinct from normal human discourse [45], and study user interactions with conversational agents [22, 25, 195]. Wizard of Oz efficiently creates the functionality of a product before it is refined via testing [110, 203].

Choosing Wizard of Oz studies helps us investigate the design space of PairBuddy before starting the implementation, as creating a fully functional conversational agent requires: 1) understanding the domain of pair programming conversational agents, 2) collecting pair programming data used to train the machine learning models of such an agent, and 3) designing and implementing a custom conversational agent architecture including the application of multiple machine-learning algorithms [69]. Additionally, platforms for developing conversational agents such as IBM Watson Assistant [191], SAP Conversational AI [168], and Oracle Digital Assistant [192] cannot be used due to their focus on simple/basic enterprise problems such as answering questions and solving tasks based on web or enterprise data.

In the words of human-computer interaction expert Jef Raskin, *"Once the product's task is known, design the interface first; then implement to the interface design."* Therefore, we used the Wizard of Oz paradigm to simulate PairBuddy's interface and interactions as we explored the design space of pair programming conversational agents. For the remainder of this paper, we use the term "PairBuddy" to refer to "the wizard's simulation of PairBuddy."

Our user-centered approach involved the exploration and evaluation of PairBuddy's design using two iterations of Wizard of Oz studies with programmers. For the first iteration, we conducted an exploratory pilot study with six university students focused on the initial design of PairBuddy's interface and interactions. For the second iteration, a larger study of fourteen university students and professional programmers evaluated a variety of design decisions informed from the pilot study and existing research.

3 PILOT STUDY (ITERATION 1)

The pilot study provided valuable feedback from programmers for the initial design of PairBuddy. The insights gathered from the pilot study served as a first step toward the exploration of the design space of pair programming conversational agents as well as a starting point for future iterations.

3.1 PairBuddy Design (Iteration 1)

PairBuddy's initial iteration primarily served to study its interface and interaction design, so we focused our research review on literature that pertained to agent design, human-robotic interactions, embodiment, software engineering, management science, intelligent tutoring systems, education, cognitive science, psychology, and gender-bias in agent design. Table 1 lists the design decisions made for PairBuddy's interactions and features in the pilot study, and are as follows:

Design: Creating the Interface and Interactions

PairBuddy was designed to include anthropomorphic characteristics in its interface and interactions.

(A) ***Interface – Embodiment via Avatar, Gender, Voice, and Text:***

PairBuddy communicated with participants by means of avatar, voice, and text to enhance human-computer interaction. An avatar was incorporated in the design because avatars make the interface more human [179] and improve understanding, engagement, and trust in novice programmers [119, 179, 209]. Furthermore, embodiment through avatars can facilitate non-verbal communication in order [19, 32] to help maintain effective pair programming relationships. Agents with avatars are given more personality attributes than those without them [179], but at the cost of heightened expectations [119]. They are particularly important to establishing first impressions [21]. Therefore, we formulated design decision `F1`: PairBuddy will be embodied by a 3D avatar.

The inevitable gendered attributes spawned from conversational agent embodiment are the target of similar gender-biases present in the real world. Particularly, female conversational agents are more often the target of negative stereotypes, sexual attention, and profanities than male agents [26]. However, they're also more likely to be forgiven, even if satisfaction ultimately doesn't change [184]. Additionally, female programmers have voiced concerns about pair programming with male partners [38] due to the expectation of being stereotyped for their gender. To analyze potential gender preferences and bias, we formulated design decision `F2`: Participants can toggle the agent between the two most common genders. While research has looked at gender bias within conversational agents and pair programming separately, we combine the analysis by evaluating participants' gender preferences for a pair programming conversational agent.

The choice between voice or text for an agent's interface depends on the use case. While text-only interfaces are less intrusive (e.g., website support), research [21, 31] has shown that voice interfaces increase users' trust in conversational agents. Additionally, cognitive science research has demonstrated that simultaneously sharing the same sensory modality for both short-term memory and active-use negatively affects response and accuracy rates under high-load conditions [187]. These findings suggests that using audio responses can help reduce programmers' cognitive load since it does not share the same sensory modality as viewing code. Additionally, the absence of a text chat saves additional screen space and reduces context switching between applications. For our design decisions F3 and F4, we explored input modality by supporting both methods of communication, allowing messages to arrive to/from the agent via both voice and text.

(B) ***Interactions – Indirect Driving, Timed Feedback, and Adapted Skill:***

In effective pair programming, both partners directly contribute to the code as drivers. However, if PairBuddy makes a mistake and overwrites code, it could become difficult for participants to undo. Since a prominent principle of HCI is that users should remain in control of their work [7, 14, 204], we formulated design decision I1: As a driver, PairBuddy will make indirect contributions through text messages so that participants can reference and modify the code themselves.

Feedback has substantial impact on learning and achievement [85, 86, 104, 114]. Among its influential properties is the timing of the feedback. Psychological research suggests that feedback on difficult concepts should be delayed, while feedback on simple concepts is more beneficial when immediate [24, 107, 159, 181, 182]. For example, interactive development environments (IDEs) already provide instant feedback via error highlighting, but when a programmer uses a misguided approach, delayed feedback is preferred to allow them time to evaluate the feasibility of their ideas. Therefore, we formulated design decision I2: PairBuddy will provide timed feedback [14]. With this decision, we hope to minimize unnecessary interruptions.

To balance pair programming roles, PairBuddy would adapt to each participant's skill-level. Just as PairBuddy seeks to replace a human pair programming partner, Intelligent Tutoring Systems (ITS) look to elicit the same effects as a human tutor [43, 59, 186]. ITSs have shown to increase student performance, and can even outperform human tutoring [180]. Like PairBuddy, Vizcaino et al. [193] designed HABIPRO, a simulated student and co-learner, to teach computer science. While HABIPRO did not pair program, it guided students' behavior by utilizing a learner model [35] to represent and track students' knowledge and progress. In the same way that ITSs adapt to learners' knowledge, we formulated design decision I3: PairBuddy's skill level will adapt to each participant such that contributions remain balanced [14]. Pair programming research informs this decision since programmers prefer when their partner is equally or more competent [128].

Programmer: Integrating Technical and Soft Skills

PairBuddy will imitate the characteristics of a programmer. Programmers' technical and soft skills are crucial for being effective team members [10, 44, 137, 214] and are used by managers to make hiring decisions [118, 122, 132, 157, 164]. Hence, we integrated both types of skills to reflect a programmer's capabilities.

(A) ***Soft Skills – Greeting and Motivation:***

While greetings vary between cultures, humans introduce themselves to make their presence known and to start conversations. Similarly for agents, Kahn et al. [97] identifies "The Initial Introduction" design pattern where agents use scripted, conventional introductions to recognize and inquire about another. It is an important design choice in human-robotic interaction, as it allows a deepening of relationships while removing initial awkwardness. Based on this researched design pattern, we formulated design decision S1: PairBuddy will introduce itself and greet the participant [14].

Motivation is seen as a driving force that has a substantial impact on a programmer's performance and productivity [13, 34], and comes from either intrinsic or extrinsic sources [34, 49]. Designing

Table 1. Design decisions evaluated in the pilot study.

ID	Design Decision	Description	Example Sources
Interface			
F1	Avatar	Embodied by a dynamic 3D avatar	[19, 21, 32, 119, 179, 209]
F2	Gender	Gender can be toggled	[26, 38, 184]
F3	Voice	Communicate via voice synthesis	[21, 31, 187]
F4	Text Chat	Communicate via shared text chat	
Interaction			
I1	Indirect Driving	Send code via text chat	[7, 14, 204]
I2	Timed Feedback	Feedback at appropriate time	[24, 104, 107, 114, 159]
I3	Adapted Skill	Balance contributions with partner	[14, 43, 59, 186, 193]
Soft Skills			
S1	Greeting	Introduce itself	[97]
S2	Motivation	Encourage, recognize, comfort, commend	[13, 34, 49, 63]
Technical Skills			
T1	Write/Feedback Tests	Generate test cases & feedback	[12, 58, 126, 127, 131, 147]
T2	Write Code	Examples from online repositories	[100, 101, 136, 145]
T3	Guidance	Provide direction via user stories	

extrinsic motivators that synergize with intrinsic motivators requires supporting a person's sense of competence without undermining their self-determination [13]. Fischer et al. [63] found that a higher perceived probability of receiving extrinsic motivation in the form of relational rewards (e.g., praise, recognition, performance feedback) [16] often positively affected creative and innovative outcomes. Therefore, we formulated design decision S2: PairBuddy will motivate using relational rewards in the form of encouragement (e.g., *"We've got this!"*), recognition (e.g., *"I see, good idea!"*), and comforting (e.g., *"That's okay, everyone makes mistakes"*). Additionally, extrinsic motivation is most effective during the stages of the creative process that make the most meaningful contributions to the project [13]. Therefore, PairBuddy will commend success through relational rewards such as, *"I knew we could do it!"* or *"We make a great team!"*.

(B) *Technical Skills – Writing Tests/Code and Giving Guidance:*
PairBuddy can generate test cases (i.e., a code fragment that specifies inputs and expected results to verify compliance with a requirement [1]) using search-based techniques and requirement artifacts. Search-based software testing uses a variety of search algorithms [126, 127, 131] to determine the most efficient path through source code that maximizes code coverage for automatic test case generation [12]. Additionally, research has demonstrated the feasibility of converting requirement artifacts such as user stories (i.e., a description of a requirement from a user's perspective [2]), acceptance criteria (i.e., the boundaries of a user story [1]), and scenarios (i.e., step-by-step description of a series of events [1]) into test cases [58, 147]. Based on this research, we formulated design decision T1: As a driver, PairBuddy can generate test cases automatically. As a navigator, PairBuddy can give feedback and answer programmers' queries based on the generated solutions. For example, PairBuddy could offer help by asking, *"Would you like me to generate a test case?"*

To further PairBuddy's competency as a driver, we formulated design decision T2: PairBuddy will provide example code from online repositories (e.g., GitHub [3]), question & answer forums (e.g., Stack Overflow [6]), and package documentation. For example, PairBuddy might send sample

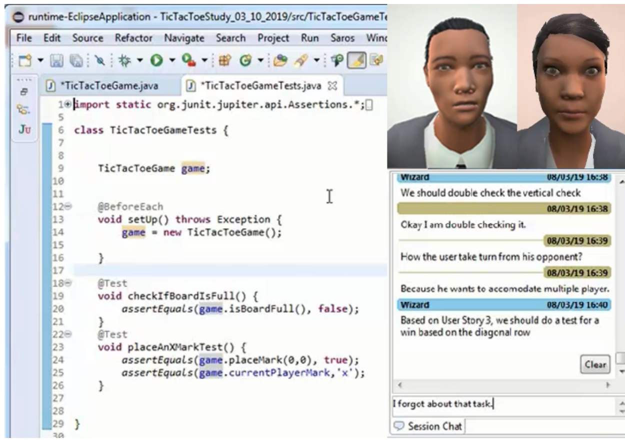

Fig. 2. The pilot study's interface including the Eclipse IDE, text chat, and avatar. Note that each participant only saw one avatar.

code from GitHub into the text chat and ask, *"Is this code example from online useful?"* Code-querying and semantic similarity algorithms allow for the collection and querying of online code repositories. Past research [100, 101, 136, 145] identifies techniques to search online repositories for semantically similar code. However, these algorithms are not always perfect, so to simulate a realistic implementation, PairBuddy's code recommendations were not an exact match with the task.

To allow PairBuddy to guide participants, we formulated design decision T3: PairBuddy will use user stories as a basis to track and direct the current objective, encouraging participants to reference them when determining how to proceed. Additionally, PairBuddy might need to ask the participant, *"What user story is that?"* when the current objective is unclear from PairBuddy's perspective. This clarifying dialogue serves a second purpose: participants are forced to refocus on the user stories, and reorient themselves toward the task's goal.

3.2 Wizard/PairBuddy Implementation

The wizard's interface and implementation integrated the aforementioned design decisions into the wizard's script and interface for the pilot study, and are detailed as follows:

PairBuddy's interface was implemented within the Eclipse IDE [54] using a modified version of the Saros plugin [56], allowing both voice and text communication. Directly integrating these features allowed participants to seamlessly interact with PairBuddy. The wizard used Saros to monitor the participant's code, but made code contributions via the text chat. PairBuddy was embodied by a 3D avatar via FACSvatar [60]. Using custom networking code, FACSvatar mapped the wizard's facial animations onto PairBuddy's avatar from a remote location, while PairBuddy's voice was generated using Google Text-to-Speech [80]. The wizard secretly monitored the participant's webcam and microphone using FFmpeg [62]. While participants were free to change the arrangement of the IDE and avatar windows, none of them did. Figure 2 shows a screenshot of the participant's interface, including the Eclipse IDE and either the man or woman avatar.

The wizard also simulated the back end design of PairBuddy, including common components of conversational agent architecture [69] such as intent classification [9, 53, 116, 123, 146, 207], dialogue state tracking [102, 111, 150, 215], dialogue policy [88, 183, 205, 211], and natural language generation [81, 149]. For example, if the participant said, *"This is why I hate coding!"*, PairBuddy

Table 2. Demographics of the pilot study participants.

P#	Age	Gender	Education	Programming Experience
PG1	24-29	Man	PhD	4+ years
PG2	30-40	Man	PhD	4+ years
PG3	24-29	Man	Masters	4+ years
PU4	19-23	Man	Undergrad	<1 year
PU5	19-23	Woman	Undergrad	<1 year
PU6	19	Man	Undergrad	<1 year

would understand their words using text-to-speech, classify the intent as "Negative Feedback," and log their difficulties in the dialogue state tracker. In response, PairBuddy would decide to "Give Motivation" based on its dialogue policy, and respond with *"Don't give up, you are almost there!"* using natural language generation. The wizard adhered to a script, which served to simulate the robotic nature of conversational agents, and contained a limited selection of dialogue for the wizard to use. Dialogue was templated according to Shneiderman's guidelines [163] and Neilsen's heuristics [135]. Since communication styles differ by gender [108], we provide gender-inclusive language [61, 130] in our script including non-authoritative suggestions [161] to both engage and motivate programmers.

The wizard exhibited a realistic level of intelligence. If participants asked questions beyond the protocol, the wizard answered, *"I'm afraid logic isn't my strong suit,"* to simulate the behavior of a potentially automated system [14]. However, to maintain participants' trust and engagement, we designed PairBuddy to give alternative contributions if it could not directly answer queries. Furthermore, the script was designed to vary the responses for the same intent. For example, motivational scripts included, *"We've got this!"* along with five or more such dialogue templates so participants wouldn't receive repeated phrases from PairBuddy. In general, the script served to simulate the back end of a fully functional conversational agent, including its limitations through the use of dialogue templates. One researcher simulated PairBuddy as the wizard in the pilot study.

3.3 Participants

Seven students majoring in computer science were recruited from our university. However, we only report data from 6 participants (3 undergraduate and 3 graduate) as one participant did not interact with PairBuddy during the study. Both undergraduate and graduate students were selected for the study to account for the varying diversity of programming skill. Participants with basic object-oriented programming experience were chosen on a first-come-first-serve basis. Upon completion, participants were given $20 in Amazon gift cards. Table 2 shows the demographics of our pilot study participants. These participants are referred to as PU# and PG# for undergraduate and graduate students respectively. For example, PU4 is the fourth undergraduate participant of the pilot study.

3.4 Study Design

Participants completed a background questionnaire prior to the pilot study. Before starting the task, participants watched video tutorials on the concepts used in the study including the driver and navigator pair programming roles, the think-aloud method, and test-driven development. The think-aloud method encourages participants to vocalize their thoughts and feelings [113, 160]. Test-driven development is a type of extreme programming that prioritizes the creation of test cases before implementing and refactoring code. Test-driven development evaluates participants' knowledge and enables diverse dialogue since each development stage provides a unique style of thinking.

Participants were asked to use pair programming and test-driven development alongside PairBuddy to complete an implementation of tic-tac-toe: a game where two players take turns marking spaces in a 3x3 board. Code for the board, along with three related test cases, were provided to the participants. Tic-tac-toe was selected for its simplicity, as anyone with basic programming experience could understand and implement solutions to the requirements without prior knowledge of the domain.

Participants were then instructed to create new test cases and functionality based on a given list of user stories, acceptance criteria, and scenarios. Participants wrote Java code in the Eclipse IDE [54] using JUnit [55] to implement testing. The duration of the task was fixed to 50 minutes to prevent participants from fatigue and to ensure that the entire study session lasted under 90 minutes.

A semi-structured interview was conducted using a script, and individualized questions explored study-specific events.

3.5 Data Analysis

The video, audio, and interviews from the pilot study were transcribed and analyzed qualitatively to evaluate the usability of PairBuddy. We used grounded theory [74] to create a codeset of the types of contributions PairBuddy made as is shown in Table 3. Transcripts were coded by two researchers. Initially, both researchers independently coded the same 20% of the transcripts, and average inter-rater reliability was measured at 86% using the Jaccard index. The remaining transcripts were split and coded separately. Interview transcripts for the pilot study can be found [4].

3.6 Results

The first iteration of PairBuddy allowed us to examine the feasibility of a pair programming conversational agent. The insights and avenues for improvement are as follows:

(1) Helping Programmers

PairBuddy helped participants complete the programming task as both a driver and a navigator. Figure 3 details the contributions PairBuddy made in three separate ways: 1) PairBuddy offered

Table 3. Code set for PairBuddy's contributions.

Contribution Type	Description	Example
Direction	Guiding participants towards goals	*"Write a method... for a horizontal win."*
Domain - Help	IDE, language, or domain knowledge	*"Looks like... an import error for JUnit."*
Method - Add	Provide example code for methods	*"Is this code... useful?"*
Method - Clarify	Give knowledge about methods	*"Get the value... by accessing its coordinate."*
Test case - Add	Provide example code for test cases	*"I can generate test cases."*
Test case - Clarify	Give knowledge about test cases	*"To test... we will need to place marks."*
Bug - Identify	Identify bugs or mistakes	*"Double check the vertical check."*
Bug - Fix	Fix bugs or mistakes	*"Error is commonly caused by..."*

Contribution Source	Description
PairBuddy Alone	PairBuddy offered contributions
Human Asked	Participants prompted PairBuddy
Human Asked - Unanswered	PairBuddy couldn't help programmer

help, 2) participants asked for help, and 3) participants asked, but PairBuddy could not provide help. The contributions types include: direction, domain-related help, method/test clarification, method/test addition, and bug identification/fixing.

When participants were lost, PairBuddy provided direction through its messages. For example, when PU6 had difficulty writing a method, PairBuddy gave guidance, *"We need to write a method to check for a horizontal win."* Furthermore, PairBuddy provided help for questions about the IDE, JUnit, language, and domain (tic-tac-toe). For example, when PG3 was having difficulties using the testing suite (JUnit), PairBuddy provided help, *"It looks like we may have an import error for JUnit."*

PairBuddy contributed knowledge by clarifying test cases and methods. For example, when PU4 was unsure about how to write a test case, PairBuddy responded, *"To test for horizontal, we will need to place marks at zero zero, one zero, and two zero."* Similarly, PairBuddy helped PU5 with her method, *"If a space is occupied by a '-' then it is considered empty."*

PairBuddy provided sample code for test cases or methods in the text chat. For instance, PairBuddy offered to contribute a test case for PG6, *"I can generate test cases based on the user scenarios. Would you like me to do this?"*, and a method for PG1, *"Does this code help you out any?"*

Finally, PairBuddy provided guidance for finding and fixing bugs. When PG1 made a mistake in the vertical win method, PairBuddy commented, *"I think there is a syntax error on line 41."*

Periodically, PairBuddy failed to answer the questions asked by participants (red in Figure 3). However, PairBuddy was designed to follow-up with separate contributions if possible (blue in Figure 3). For example, when PU5 asked, *"What would the arguments be for the method to find if there is a winner?"*, PairBuddy was not designed to answer this difficult type of question, so instead, it made a suggestion from online, *"Is this code example from online useful?"*

(2) Effect of Programming Experience

Figure 3 shows the trend that graduate students (PG1 - PG3) interacted with PairBuddy during the second half of the study, while undergraduate students (PU4 - PU6) interacted from the beginning. One graduate student, PG1, interacted from the start, but PG2 and PG3's first interactions were at 19:00 minutes and 17:30 minutes respectively. In his interview, PG2 said that he wanted help just-in-time and preferred to work solo on his tasks, *"I was thinking, and I want to have time for... quiet and focus."* We conjecture that experienced participants were less trusting initially, but their trust increased overtime. For example, PG3 said, *"As it went on, I was like, 'Hey,... we have the same idea for this.' "*

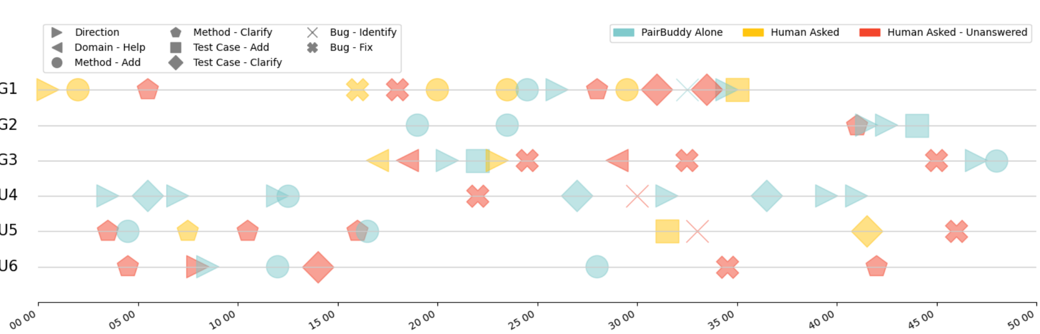

Fig. 3. A timeline illustrating PairBuddy's contributions during the pilot study. Contributions were either offered by PairBuddy (blue) or asked by participants (yellow). In some instances, PairBuddy was unable to provide help (red). Contributions types include direction, domain-related help, method/test case addition, method/test case clarification, and bug identification/fixing.

(3) User Experience

Participants enjoyed working with PairBuddy and appreciated its company. PG2 found PairBuddy's voice supportive, saying, *"Just hearing him talk helped me stay on track and stay focused on the task at hand and what needs to be done."* Similarly, PU5 indicated that she often thinks out-loud with a partner in her classes, explaining, *"I'll like talk, 'I don't know what I'm doing,' and [my friend] will be like, 'I don't either.' "* Additionally, PG3 felt synergy with PairBuddy, commenting, *"I think we work together well."*

Participants also enjoyed the motivational aspect of PairBuddy. In fact, all participants responded positively to motivation. For example, when PG1 fixed a bug that he was struggling with, PairBuddy celebrated saying, *"Yay! You did it!"*, and PG1 smiled responding, *"Thanks!"* Similarly, PU6 said, *"Thank you so much dude. Your motivation is so good. Thank you, thank you, your motivation is getting me through."* However, participants' reactions to positive feedback varied, as PG2 only said, *"Okay thanks."*

3.7 Lessons Learned

Feedback from the pilot study revealed many shortcomings of PairBuddy's design and informed modifications to the original design decisions for use in the main study. We found the following limitations of both PairBuddy's design and the study's implementation:

Avatar Did Not Support Lip-Sync: Participants rarely looked at the avatar window throughout the study. PU6 put it bluntly, *"I forgot he [the avatar] was even there."* While the avatar often went unused, PU4 noted that avatars play a specific role in communication for him, *"I'm actually hard of hearing just a little bit, but I take cues, at least for certain words, I take cues from reading lips, so [lip-sync] would actually help me a lot."* Additionally, research finds that lip-synchronization heightens the level of an avatar's embodiment [75]. However, in the pilot study, PairBuddy did not include a lip-sync feature, making communication less accessible and embodiment weaker.

Gender Toggle was Not Utilized by Participants: All six participants never changed the gender of the avatar, which limited our ability to make any explicit determinations for participants' gender preferences. Additionally, the style of interaction between genders was indiscernible due to the low participant count, causing individual differences to out-shine gender-based differences. To gather relevant gender data, we must use an approach that can discern individual gender preferences.

Participants' Bias Toward Text: When given the option to use either voice or text to message the agent, participants initially felt more comfortable using text and often forgot that voice was even an option. In fact, 5/6 participants used text rather than voice. However, when PU6 was accustomed to typing messages, he was taken off-guard when PairBuddy responded to his voice, *"Wait, can you hear me? That would be kinda cool if you can hear me."* Participants who used text forgot PairBuddy could hear them, often verbalizing their message before typing it into the chat. For example, PG1 spoke out loud, *"How to write the method?"* before sending it word-for-word in the text chat. The inclination to vocalize questions suggests that the decision to use text over voice was partially based on preconceptions and habits, rather than utility. While participants were biased toward text, they spent an unnecessary amount of time typing messages. This notion of slow text communication is supported by research on interaction speed [155], which finds that typing is 3x slower than voice recognition (on mobile devices).

Participants Needed More Guidance: Although we envisioned PairBuddy to act as both a driver and a navigator, design decision T2 transformed PairBuddy's role as a driver into a code recommendation system. Since PairBuddy couldn't directly contribute to the code, its ability to meaningfully guide progress was limited.

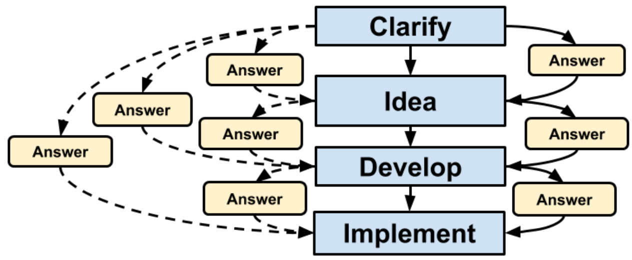

Fig. 4. Creativity stages (blue) progress from `Clarify` to `Implement`, but humans skipped the `Idea` and `Develop` stages (dashed arrows) when PairBuddy provided answers (code/test cases). Typically, human-human pairs progress through each stage in succession (solid arrows).

Interjections Become Interruptions While participants generally appreciated when PairBuddy interjected with helpful dialogue, they did not like it when interjections became interruptions. Even with PairBuddy's timed feedback (design decision `I2`), participants wished PairBuddy did not disrupt their train of thought. PG2 explained, *"When I'm trying to think about a problem... Most of the time I need a quiet place to think."* Similarly, PG1 complained, *"Some of the things were distracting to me and unexpected... That I didn't like."* While research finds that interruptions greatly reduce progress on writing assignments [64], our participants still wanted an agent that is engaging and socially present. PG3 expressed his preferences, *"I would prefer someone who is more engaged,"* and PU5 did too, *"I don't feel as crazy or lonely because it's like 'Oh, I'm talking to a computer. It's fine,' but like when you're doing it alone it's like, 'Ah, there's no one here.' "*

Lack of Support for Creativity: Upon analyzing the transcripts, we realized that participants followed creative strategies as identified in the Osborn-Parnes Creative Problem Solving Process. Osborn-Parnes Creativity is frequently used to understand the creative process of an individual [89, 99, 140, 142] and is vital to programmer success [57, 194, 212], especially when solving open-ended problems [18, 28, 112, 117, 144, 198]. Particularly, it describes how programmers step through a series of creativity stages. Figure 4 illustrates how programmers start at the `Clarify` stage of creativity, where information, goals, and challenges are identified. In the subsequent `Idea` stage, ideas regarding how to solve the challenge are generated. Then, in the `Develop` stage, solutions are evaluated, strengthened, and selected for "best fit." Finally, in the `Implement` stage, the resulting solutions are written into code. In human-human pair programming, humans follow the creativity stages in sequence [109] (solid arrows in Figure 4). However, we observed that participants skipped the `Idea` and `Develop` stage by directly copying implementation from PairBuddy's code examples (dashed arrows in Figure 4). Therefore, PairBuddy should support each individual creativity stage, specifically the skipped stages of `Idea` and `Develop`.

4 MAIN STUDY (ITERATION 2)

Through the insights gained from the pilot study and a more comprehensive research review, we improved the design of PairBuddy and conducted a second "main" Wizard of Oz study.

4.1 PairBuddy Design (Iteration 2)

The design of PairBuddy for the main study intended to imitate human pair programming to the highest degree that can be realistically achieved through the capabilities of current research. Table 4 lists the design decisions used in the main study. Those adapted from the pilot study remain white, while new design decisions are highlighted in blue. Design decision modified or added for the main study are as follows:

Design: Creating the Interface and Interactions

(A) ***Interface – Embodiment via Avatar, Gender, Voice, and Text***

For design decision F1, PairBuddy's dynamic 3D avatar was modified to include lip-synchronization. Furthermore, to explore the trade-offs of using text vs. voice for a pair programming agent, we modified design decisions F3 and F4: communication with PairBuddy will be primarily verbal, while reserving the text chat for resources such as images or links. This decision attempted to model typical human-human remote communication.

(B) ***Interactions – Direct Driving, Timed Feedback, Adapted Skill, Typing Speed, and Redirect Suggestions***

To allow PairBuddy to more effectively guide code progress as an active driver, we modified design decision I1: PairBuddy will make direct code contributions through the IDE rather than sending code snippets through the text chat, and as an active navigator, PairBuddy will provide more specific feedback. To support PairBuddy's heightened engagement, we conducted a research review of automated code and feedback generation algorithms. Since algorithms are prone to mistakes, PairBuddy did not overwrite large chunks of participants' code, and instead, commented them out.

Since PairBuddy directly contributed code through the IDE, it was necessary to consider the speed at which PairBuddy typed. Intentionally limiting an agent's typing speed might be unappealing and cause programmers to become impatient, but pasting large chunks of code might be too overwhelming to comprehend. Therefore, we formulated design decision I4: PairBuddy will incrementally paste small snippets of code to prevent programmers from becoming impatient or overwhelmed. This design decision was informed from our best reasoning rather than previous research.

In pair programming, the role of the navigator includes providing feedback and suggestions to the driver. Sometimes, the navigator performs "backseat driving" where they instruct the driver directly [94]. However, current technology does not fully support the implementation of arbitrary ideas in this way, so in the pilot study, PairBuddy was designed to immediately admit its limitations saying, *"I'm sorry, I don't know how to help with this."* Unfortunately, this design decision often marked an abrupt end to the conversation. Therefore, to encourage higher engagement, we formulated design decision I5: PairBuddy will redirect suggestions back to the programmer through dialogue such as *"How would that look like?"*, *"Do you want to try?"*, or *"Can you do that for me?"*. If the programmer insists, however, only then will PairBuddy admit its limitations.

Programmer: Integrating Technical and Soft Skills

(A) ***Soft Skills – Leadership, Uncertainty, Social Presence***

Research finds that a democratic leadership style is most effective for pair programming [108]. A way for PairBuddy to integrate democratic leadership is to practice effective pronoun use. Research has shown that leaders use collective pronouns (e.g., "we" and "us") to gain influence in a group [96], but use personal pronouns (e.g., "I" and "me") to "own up" to mistakes [148]. Therefore, we formulated design decision S3: PairBuddy will display leadership by attributing successes to the group while taking ownership for its mistakes. For example, when a test case fails, PairBuddy will take ownership, saying, *"I think I made a mistake,"* but if all the test cases pass, PairBuddy might say, *"Great, we did it!"*

Research suggests that conversational agents should preemptively use uncertainty to avoid cases where agents make large miscommunications [14, 17], and the resulting conversational breakdowns decrease users' satisfaction, trust, and willingness to continue talking to conversational agents [90, 91, 120]. Additionally, research on pair programming shows that people often ask for verification

Table 4. Design decisions of PairBuddy's main study implementation. Decisions modified from the pilot study remain white, while new additions are highlighted in blue.

ID	Design Decision	Description	Example Sources
Interface			
F1	Avatar	Embodied by a 3D lip-synced avatar	[19, 21, 32, 119, 179, 209]
F2	Gender	Gender can be toggled	[26, 38, 184]
F3	Voice	Communicate via voice synthesis	[21, 31, 187]
F4	Text Chat	Paste images or links	
Interactions			
I1	Direct Driving	Edit code via IDE	
I2	Timed Feedback	Feedback at appropriate time	[24, 104, 107, 114, 159]
I3	Adapted Skill	Balance contributions with partner	[14, 43, 59, 186, 193]
I4	Typing Speed	Paste small sections of code	
I5	Redirect Suggestions	Participants implement their suggestions	
Soft Skills			
S1	Greeting	Introduce itself	[97]
S2	Motivation	Encourage, recognize, comfort, commend	[13, 34, 49, 63]
S3	I vs. We	Share success and personalize mistakes	[96, 148]
S4	Uncertain/Verification	Show uncertainty via verification of work	[17, 108]
S5	Social Presence	Actively listen rather than interrupt	[41, 64, 84, 134]
Technical Skills			
T1	Write/Feedback Tests	Generate test cases & feedback	[12, 58, 126, 127, 131, 147]
T2	Write/Feedback Code	Generate code & feedback	[46, 100, 101, 136, 145, 213]
T3	Guidance	Provide direction via user stories	
T4	Creativity Support	Prompt divergent & convergent thinking	[92, 93]
T5	Feature Location	Locate code from a description	[115, 121, 158]
T6	Unnecessary Code	Suggest deleting unused code	[178]
T7	Missing Code	Determine where more code is needed	[73]

after each creative stage of development [108]. Therefore, we formulated design guideline S4: PairBuddy will convey uncertainty by asking for verification in order to prevent conversational breakdown. To potentially support this, the uncertainty of the PairBuddy's dialogue could be based on the confidence of the machine learning algorithms used to generate code and feedback.

PairBuddy's timed feedback from the pilot study did not eliminate interruptions, since many participants still voiced complaints. When participants were deep in thought, they wished that PairBuddy listened rather than interrupted. However, maintaining an active presence as a navigator is key to preserving the balance between roles. To this end, we utilize active listening – a technique to allow a speaker an outlet for self expression [153] – to maintain social presence in a conversation. For conversational agents, social presence is the feeling and perception of interacting with a human being [138], and has been shown to increase the use of conversational agents [134], and can even increase perceived usefulness, trust, and enjoyment [84]. Therefore, we formulated design decision S5: PairBuddy will increase its social presence using active listening through acknowledgements and confirmation questions. In this way, PairBuddy can encourage participants to think their ideas out-loud rather than interrupting their thought process through dialogues such as, *"What are you thinking?"* and *"What does this code do?"*.

(B) *Technical Skills: Guidance, Creativity Support, Feature Location, Unnecessary Code, and Missing Code*

To encourage programmers to interact more with PairBuddy (relative to our pilot study), we conducted a thorough research review to identify the potential capabilities of automated code and feedback algorithms, as well as techniques to support creative problem solving.

PairBuddy's power to provide context-specific code and feedback was dependent on the capabilities demonstrated by research. Automated code and feedback techniques [46, 213] show the ability to provide feedback on code using a dataset of past programming solutions. However, this research is either limited by the specificity of the feedback [46] or has only been demonstrated in a simple programming language (iSnap) [213]. To increase the specificity of PairBuddy's contributions, we modified design decision `T2`: PairBuddy will have the limited ability to contribute context-specific code (as the driver) and provide meaningful feedback (as the navigator) based on a dataset of past solutions. For example, if PairBuddy detects that a participant's code is semantically similar yet deviates from a known solution, it might suggest, *"There might be a mistake on line 66."* However, if the participant writes code that deviates from all past solutions within the database, PairBuddy will be unable to provide guidance.

Since PairBuddy attempts to replace a human partner, it will be designed to integrate the creative problem solving stages used by humans. However, in the pilot study, PairBuddy only supported the `Clarify` stage (via direction from user stories) and the `Implement` stage (via example implementation from online). Therefore, we formulated design decision `B4`: PairBuddy will add support for the remaining `Idea` and `Develop` stages by using concepts from Idea Garden [92, 93]. Idea Garden suggests the use of probing questions to promote a programmers' use of diverse problem-solving strategies. We integrated two strategies into PairBuddy's script: working backwards from the goal and encouraging divergent thinking before convergent thinking. For example, PairBuddy might ask, *"What are all the possible ways we could do this?"*, *"Why do you think so?"*, or *"What data structure could we use?"*. As a navigator, PairBuddy can prompt the participant to move from the `Idea` to the `Develop` stage by asking, *"What would that [idea] look like?"* As a driver, PairBuddy will provide the empty structure of the code to help programmers conceptualize the solution. Additionally, PairBuddy can discuss the general structure of the code saying, *"I think we should use a for loop/if statement here,"* or *"Should we use a while loop or a for loop?"*

Both static [121] and dynamic [115] techniques exist to automatically search source code for specific features or descriptions. For the Java programming language, Flat3 [158] is an Eclipse plugin that searches source code based on arbitrary descriptions (e.g., "file saving"). Based on feature location algorithms, we formulated design decision `T5`: PairBuddy can identify locations within the code that match a description. For example, if the participant asks, *"Do we return false in the tie game detection function?"*, PairBuddy can identify the *isTied()* function as the context to the question.

Many programming IDEs include refactoring tools that automatically detect uncalled functions or unused variables. For the Java programming language, UCDetector [178] is an Eclipse plugin that can identify unused variables, functions, and classes. Based on this functionality, we formulated design decision `T6`: As a navigator, PairBuddy can detect unnecessary blocks of code and suggest that the programmer consider deleting or modifying them. For example, PairBuddy might suggest, *"I believe we have unnecessary code in the tie test function."*

Research on automated feedback has shown the ability to automatically determine the locations where code is missing based on pre-defined functions in Haskell [73]. Since Haskell is a functional programming language, this research does not necessarily apply to an imperative language like Java. Regardless, the supporting research of previous design decisions [12, 46, 127, 131, 213] would

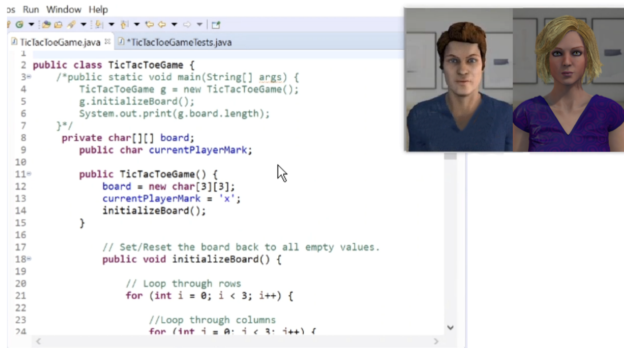

Fig. 5. The main study's interface including the Eclipse IDE and the man/woman avatar. Note, each participant only saw one avatar at a time.

be potentially capable of identifying missing code, so we formulated design decision T7: As a navigator, PairBuddy will direct the driver's attention to locations where more code is needed.

4.2 Wizard/PairBuddy Implementation

The wizard's interface and implementation integrated PairBuddy's new design decisions as discussed in Section 4.1. The interface design of PairBuddy included the following changes from the pilot study:

The Saros plugin for the Eclipse IDE was used for remote collaboration rather than to simply view the participant's code. Skype [173], Discord [170], and Google Hangouts [172] facilitated audio and video communication with the participant. To support the integration of lip-synchronization for PairBuddy's avatar (based on the lessons learned from the pilot study), we utilized the Facerig [176] avatar embodiment software. From the limited selection of Facerig avatars, we chose two avatars (man and woman) based on their professional appearance. Figure 5 shows the interface that participants used for the main study.

The wizard's script for the second iteration of PairBuddy was enhanced to include the new design decisions discussed in Section 4.1. Two researchers acted as the wizard to simulate the behavior of PairBuddy. The first researcher focused on verbal and non-verbal communication, while the second researcher focused on gathering and providing code for use as the driver. The decision to have two wizards was informed from the pilot study, since our single researcher had difficulty balancing all of their responsibilities as PairBuddy.

To quicken the wizard's response speed, a custom Electron [175] application was created to provide the researcher a simple user interface to select dialogue templates, provide context-specific information, and generate audio via Google Text-to-Speech [80]. The generated voice was sent to Voicemeter [177] and subsequently Facerig to perform lip-synchronization, and the resulting avatar and voice streams were sent to participants via communications tools (i.e., Skype, Discord, Google Hangouts).

4.3 Participants

Due to the COVID-19 pandemic, study sessions were conducted remotely, and recruitment of participants was done via snowball sampling, social media (Facebook [171], Twitter[174]), and hiring sites (Upwork [67]). Our recruitment approach helped collect participants from universities and industries across the country, in addition to local participants. We recruited 14 participants on a first-come first-serve basis, including 8 students (4 men / 4 women) and 6 professionals (3 men /

Table 5. Demographics of the main study participants.

P#	Age	Gender	Education	Programming Experience
MS1	19-23	Woman	Undergrad	3 years
MS2	19-23	Man	Undergrad	2 years
MS3	19-23	Man	Undergrad	2 years
MS4	19-23	Woman	Undergrad	2 years
MS5	19-23	Woman	Undergrad	3 years
MS6	19-23	Man	Undergrad	3 years
MS7	19-23	Woman	Undergrad	2 years
MS8	19-23	Man	Undergrad	4+ years
MP9	30-40	Woman	Masters	4+ years
MP10	41+	Man	Masters	4+ years
MP11	19-23	Man	Undergrad	3 years
MP12	30-40	Woman	Masters	3+ years
MP13	30-40	Man	Masters	2 years
MP14	19-23	Woman	Undergrad	1 year

3 women). We purposefully achieved a gender balance since research has shown a difference in preference and behavior across various genders [30, 36, 71, 125, 129, 162]. Note, all participants self-identified as men and women in their background questionnaires, so we report results from these two genders. We refer to participants of the main study as either MS# or MP# for students and professionals respectively. For example, MS4 is the fourth student participant of the main study. Student participants were given $20 and professionals were given $40 in Amazon gift cards.

4.4 Study Design

The study design remained the same as the pilot study except for the following changes:

As previously mentioned, the main study was conducted during the COVID-19 pandemic, so all study sessions were conducted virtually. Prior to this 40 minute study, participants completed a self-efficacy questionnaire. The main study included a tutorial that explained PairBuddy's abilities and encouraged participants to use PairBuddy while completing their task. This introduction attempted to combat the low level of initial (and total) interactions with PairBuddy in the pilot study, and aimed to ensure that all participants started with the same preconceptions of PairBuddy's abilities [14]. Additionally, a pre-study questionnaire was used to establish a baseline for participants' self-efficacies using a 7 point Likert scale [95] for a maximum score of 63 points.

The study was designed to ensure that each participant interacted with both of PairBuddy's gender embodiments. Rather than evaluating gender preferences directly, we used a within-study design where the avatar's gender was changed halfway through the study to compare preferences on an individual basis. The initialization of PairBuddy's gender was evenly distributed with the genders of the participants.

The study was followed by another self-efficacy questionnaire, a pair programming preference questionnaire, and interview questions to triangulate our study findings. The same interview questions used in the pilot study were integrated with additional questions related to the usability of the features that were more difficult to evaluate using our think-aloud study.

4.5 Data Analysis

The resulting transcripts were analyzed using the codeset identified in the pilot study (Table 3). Three researchers independently coded 20% of the transcripts and reached an agreement on 93% of the coded data by calculating inter-rater reliability using the Jaccard index. The wizard's script, study transcripts, and interview transcripts can be found [4].

4.6 Limitations

One limitation of the main study is its small sample size of 14 programmers. Although our sample was gender-balanced and included a wide range of skill levels (8 students and 6 professionals), the study's small size did not allow us any room for further stratification. On the other hand, our study helped evaluate PairBuddy's usability with a diverse population of programmers. While we only studied one programming language (Java) and one IDE (Eclipse), our focused approach was an appropriate choice to show baseline feasibility. Studies with more diverse programmers and languages will need to be conducted in the future.

Programmers' experience with the Java language, test-driven development, and pair programming may have affected their experience with PairBuddy. For example, PairBuddy provided very few contributions for our most experienced professional, MP10, while providing more contributions to other less-experienced professionals. In the future, more variables should be considered in our analysis.

Furthermore, our results are based on the simple task of tic-tac-toe, but the game's simplicity may have affected our usability results since one participant mentioned that he might not trust PairBuddy for more complex tasks. While the variation of task complexity needs to be further explored with additional studies, nonetheless, we believe that our task was a good representation of student assignments and that our results confirm the feasibility of PairBuddy in educational settings.

We studied the usability of PairBuddy in a virtual lab setting for only 40 minute sessions. Although this was good starting point, our study could not provide a long-term perspective regarding the usability and user experience of programmers at this moment. For instance, will interactions with PairBuddy for longer periods of time increase or decrease trust?

Finally, we were forced to conduct a virtual lab study due to COVID-19, which may not mimic traditional lab studies where participants sit in a controlled lab environment. However, we view this as an opportunity to evaluate the usability of PairBuddy in a hybrid lab study consisting of both real-world and controlled lab settings.

4.7 Results

The second iteration of PairBuddy served as a realistic portrayal of a pair programming conversational agent. Insights collected are as follows:

(1) Helping Programmers

The results from Figure 6 show that participants were most likely to request help from PairBuddy (yellow) during the second half of sessions. We conjecture that the change in requests overtime is the result of participants' initial distrust in PairBuddy, as noted by MS5's comment, *"I think I was a little distrustful at first half of the application, but then after working with it, especially after I saw that it was helping me solve the problem... I trusted it a little more."*

Participants' confidence in their coding abilities increased with the usage of PairBuddy. On average, participants' self-efficacy scores increased +3.64 from a total of 63 points (49.71 to 53.07) after their interactions with PairBuddy (Table 6). Only 3/14 participants reported a decrease in self-efficacy. MS6 was by far the largest outlier, with a difference in self efficacy of -18 points: 13

Table 6. Self-efficacy scores for main study participants from pre- and post-study questionnaires. Responses to nine questions were scored on a seven point Likert scale for a maximum score of 63 points.

P#	Pre-Study	Post-Study	Difference
MS1	50	56	+6
MS2	57	61	+4
MS3	50	45	-5
MS4	52	58	+6
MS5	41	56	+15
MS6	46	28	-18
MS7	43	55	+12
MS8	55	51	-4
MP9	48	58	+10
MP10	51	54	+3
MP11	60	61	+1
MP12	44	53	+9
MP13	50	50	0
MP14	49	57	+12
Average	49.71	53.07	+3.64

points fewer than any other participant. Otherwise, 10/14 participants saw an increase in their self-efficacy.

Getting Directions and Domain-Related Help: Participants received direction from PairBuddy as seen in Figure 6. PairBuddy's guidance helped start participants in the right direction, as PS5 explained, *"It definitely helped me get started."* Participants received guidance for an unfamiliar language, including MP11 who said, *"Getting started was the hardest part for me, like trying to wrap my head around Java again."* Some learned new techniques, like PS2 who commented, *"How to do test-driven development... the robot helped me make sure that I was writing it correctly, and I liked that."*

Clarify a Method or Test Case: Participants mentioned that PairBuddy helped clarify task objectives, including MP9 who commented, *"I felt like PairBuddy was pretty good at understanding the overall objective of the of the project."* This positive feedback was largely caused by PairBuddy's ability to `Clarify` the task based on user stories or generated solutions. For example, PairBuddy provided insight when MS6 was trying to finish a method, *"I think we are missing code in this function."* However, PairBuddy often couldn't answer `Idea`-related questions, so when MP14 asked, *"Do we need to account for the opposing player mark?"*, PairBuddy reflected the question by responding, " *What do you think?"* While such responses continued the conversation, they did not provide the direct, human-like answers that participants desired.

Adding Methods or Test Cases: As the driver, PairBuddy wrote both test cases and method functionality. PairBuddy's contributions helped participants make progress, and almost all participants ended up reusing the code given by PairBuddy. This includes MP11, who successfully adapted PairBuddy's test case for a vertical win into both horizontal- and diagonal-win test cases. For some participants, PairBuddy's code offered a new approach. For instance, when MS1 struggled, PairBuddy's alternative suggestion provided the insight necessary for MS1 to complete her vertical win functionality.

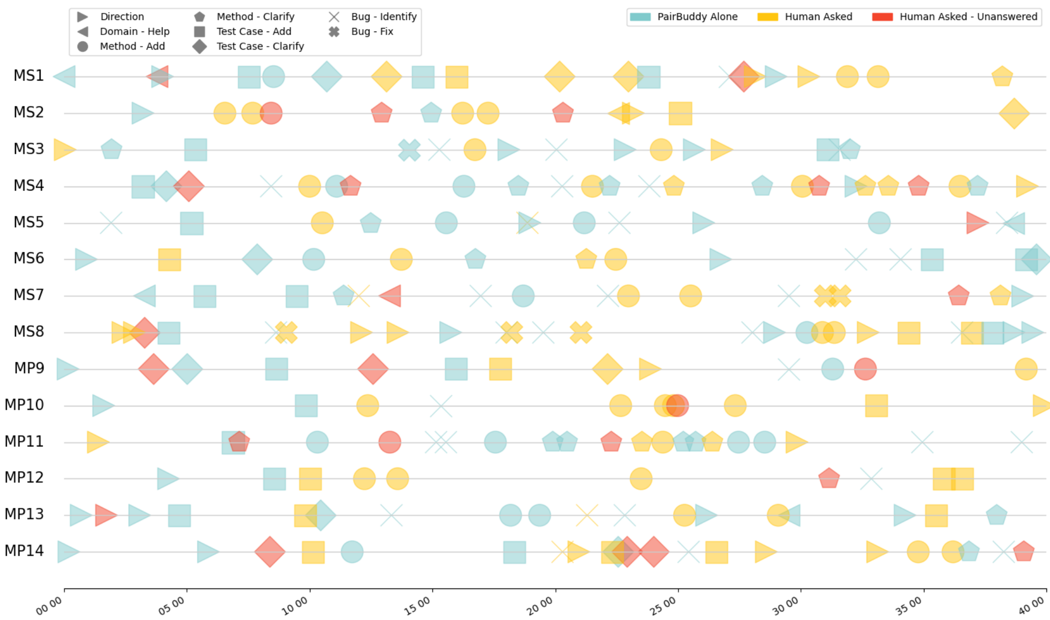

Fig. 6. A timeline illustrating the contributions PairBuddy made in the main study. Contributions were either offered by PairBuddy (blue) or asked by participants (yellow). In some instances, PairBuddy was unable to provide help (red). Contributions include direction, domain-related help, method/test case addition, method/test case clarification, and bug identification/fixing.

Identifying or Fixing Bugs: As both the driver and navigator, PairBuddy helped participants find and fix bugs, leading to an increase in code quality. As the navigator, PairBuddy would often point to locations of errors or provide hints. For MS1, PairBuddy pointed, *"I think there is a typo on line 107,"* leading her to immediately fix the mistake. On some occasions, PairBuddy had to fix bugs itself when participants struggled to understand the error that PairBuddy was referring to. When MP13 had trouble fixing an identified mistake, PairBuddy took action by asking for permission to drive and fixing the error while completing the remainder of the method.

When PairBuddy Failed to Answer: Figure 6 shows the instances where PairBuddy was unable to help participants (red). Either PairBuddy would redirect queries (e.g., *"What do you think?"*) or admit its limitations (e.g., *"Sorry, I'm not good at logic"*). PairBuddy's inability to provide help like a human negatively affected some participants' trust and confidence in PairBuddy's abilities. After his queries went unanswered three times, MS2 mentioned PairBuddy's inaction as a negative aspect in his interviews, *"He couldn't respond to that. At least not in a way that made sense."* Participants expressed concerns about PairBuddy's inability to answer "why" questions or give reasoning behind the code it write. After PairBuddy failed to help, MP9 later commented in her interviews, *"I felt like they [PairBuddy] knew what to do. But they weren't always able to communicate to me the why."*

(2) Usability of PairBuddy

In their interviews, participants expressed mostly positive experiences with PairBuddy (Table 7):

Enjoyed/Helped/Learned:

Enjoyed: A vast majority (12/14) of participants enjoyed working with PairBuddy. MP10 was the most enthusiastic commenting, *"It is really, really awesome."* PairBuddy's design surprised many participants, including MS5 who said, *"I thought it's really cool technology. I didn't know stuff like that was out there, really. So it's pretty novel."* The only participant who disliked PairBuddy was MS6, who mentioned, *"Not particularly...There wasn't as much human interaction."*

Helped: Most participants (13/14) expressed that PairBuddy helped them solve the task. MS8 thought PairBuddy saved him time saying, *"It did, yea. It saved me some time because a lot of that time would have been spent doing trial and error."* Even MS6, who disliked PairBuddy, indicated that it helped him solve the problem, *"[PairBuddy] helped me understand... how to start approaching things so that I could start going myself."*

Learned: However, only 6/14 participants said that they learned from PairBuddy, and 3/14 remained neutral. Some participants mentioned that they learned task-related concepts, including MS6 who commented, *"I think learning the general structure of what pair programming is,"* while others learned new creative strategies including MS3 who said, *"I got it like a new perspective... I guess I still need to start thinking more outside the box."* On the other hand, 5/14 responded they did not learn from PairBuddy. Unlike a human, PairBuddy didn't explain the code it wrote, as MS3 described that PairBuddy *"left it up to [their] interpretation."* PairBuddy's inability to express its reasoning may have contributed to lower satisfaction with PairBuddy.

`Future Design`: To improve PairBuddy's design going forward, we will explore more ways to impart knowledge through code and test case generation to the extent that current research allows (details in Section 5).

Trusted: Most (12/14) participants trusted PairBuddy overall (Table 7). Participant's trust may have been based on the assumption that, *"[PairBuddy] was familiar with what the program should look like,"* as described by MS7.

Navigator and Driver Roles: PairBuddy was trusted as a navigator (12/14) more than as a driver (8/14).

PairBuddy acted as an *active navigator*, using soft skills (leadership, motivation, uncertainty, and social presence) to help increase participants' confidence and trust. 12/14 participants indicated trust and confidence, as confirmed by MS2's comment, *"I thought it [navigation] was it's [PairBuddy's] most useful function."* Similarly, MS6 mentioned, *"One thing that was helpful is that when they made their navigator suggestions, like all of them seemed reasonable."* The motivation PairBuddy provided was praised by participants like MS7 who explained that, *"It was cute that it said, 'Good job,' and gave positive affirmations."* PairBuddy's social presence provided a sense of security toward participants' work, as evidenced by MS2's comment, *"I think that I can keep on coding and not like, worry about checking it three more times because PairBuddy's got my back."* MP9 treated PairBuddy as someone to talk to, commenting, *"I could say things in the same way that I would to a human."*

PairBuddy acted as an *active driver*, using technical skills to make contributions, give just-in-time feedback, and support divergent thinking. Participants appreciated PairBuddy's divergent thinking including MS5 who commented, *"I'm gonna approach coding in the future... [by] stepping back and looking for alternate approaches."* Although participants trusted PairBuddy's code and just-in-time feedback, they wanted even more assistance. This desire was evidenced by MS3's suggestion, *"Maybe have it give more frequent and more pointed advice on the code itself."* MS7 wished PairBuddy's feedback was more in-line with their own ideas, *"I think my trust would decrease if I noticed that the suggestions it was giving me weren't like really aligned with what I was trying to do."*

Preference for PairBuddy vs. Human: Participants had mixed opinions when asked to compare PairBuddy with a human. Some participants preferred PairBuddy's collaboration style including MS5 who commented, *"I think I was getting feedback from it akin to what I would from humans, and in fact, [it] is a little less intrusive than some humans are when it comes to how I'm writing my* *code.*" However, many participants wished to work with humans to discuss ideas, since PairBuddy couldn't explain its own. In fact, participants ignored PairBuddy's suggestions a total of 16 times and often chose to pursue their own ideas. MS3 wished PairBuddy provided explanations as a driver, and described that, *"The code that it [PairBuddy] wrote was functional for what it did, but it [PairBuddy] never actually explained what it drove."* Participants also expected PairBuddy to offer more contextual feedback, as MS6 mentioned, *"If I were to ask a human, they wouldn't just say 'I may have made a mistake,' they would explain what they did and explain why they were thinking they made a mistake or why they thought it wasn't a mistake."*

`Future Design`: Going forward, we will improve PairBuddy's ability to generate explanations and contribute to discussions as much as current technology will allow (details in Section 5).

Embodiment: The embodiment of PairBuddy's voice was much appreciated, while the presence of PairBuddy's avatar received mixed feelings. Feedback on the presence of the text chat leaned negative (see Figure 7).

Avatar: Most participants preferred (6/14) or were neutral (6/14) towards the presence of an avatar. Some participants found it natural to interface with an avatar, as described by MS7, *"It feels more natural to talk to it than if there's no avatar."* Similarly, MS4 felt a closer connection to PairBuddy when personified, saying, *"[I felt connected] more so than if it was just a voice... or just text."* On the other hand, 2/14 participants did not like the avatar. These differing opinions coincide with mixed empirical evidence for the necessity of avatars for agent embodiment [50, 83, 87, 133, 185, 208].

Voice: Voice was PairBuddy's most desired feature (13/14 preferred, 0/14 not preferred). The ability to communicate with PairBuddy verbally was very intuitive for participants. MP14 particularly enjoyed being able to think out loud, commenting, *"I liked being able to speak... [so that PairBuddy can] understand what I'm saying while I'm talking out loud rather than having to type out everything."* MS2 found PairBuddy's voice to be less distracting when he was deep in thought saying, *"I feel like the voice is better when you're focused on programming."*

Text: The desire for text messages was mixed (2/14 preferred, 7/14 not preferred). Participants who advocated for text suggested it as an addition, rather than a replacement, of voice communication. MS5 suggested to display PairBuddy's messages as text bubbles, explaining, *"I also could also see like a little pop up in the corner maybe being helpful."* However, most participants thought text would be distracting, as MP9 described, *"I wouldn't want to be context switching between the programming and the messages."* These results contradict preferences for text chat in the pilot study, and may be influenced by the lack of text messages in the main study.

Gender: Participants' gender preferences were identified by asking whether they preferred the first or second avatar/voice. Many participants' reasoning for their choice was not gender-related; MP13's reason was *"just because I started with it,"* while MS4's was because *"she sounded a little less robotic."* While previous research has identified gender preferences and bias toward conversational agents [26, 38, 184], no useful patterns emerged from our data that evidenced any effect of PairBuddy's gender.

`Future Design`: We will keep the same embodiment design with additional choices for PairBuddy's gender, ethnicity, and accompanying voice tones.

Tone/Style/Feedback: Participants preferred a casual tone, had mixed opinions of PairBuddy's dialogue style, and desired non-neutral feedback (Table 7).

Polite and casual tone: Most participants preferred if PairBuddy used a casual (8/14) over a polite tone (2/14). Since PairBuddy's tone leaned more casual, all participants were satisfied with PairBuddy's tone (14/14). Even participants who preferred polite dialogue liked PairBuddy's tone the way it was. In his interview, MP10 commented, *"I felt it was excellent. It didn't feel to be like trying*

Table 7. Responses to interview questions in the main study.

Interview Question	Yes	Neutral	No
Did you enjoy working with PairBuddy?	12	1	1
Did PairBuddy help you solve the problem?	13	0	1
Did you learn anything from PairBuddy?	6	3	5
Did you trust PairBuddy?	12	2	0
Did you trust PairBuddy's navigation?	12	1	1
Did you trust PairBuddy's navigation over a human?	5	6	3
Did you trust PairBuddy's driving?	8	5	1
Did you trust PairBuddy's driving over a human?	4	4	6
Was PairBuddy's avatar helpful?	6	6	2
Was PairBuddy's voice helpful?	13	1	0
Would more text have been helpful?	2	5	7
Do you prefer a casual over a polite agent?	8	4	2
Was PairBuddy too casual (yes) or too polite (no)?	0	14	0
Do you prefer human-like dialogue over robotic?	5	6	3
Do you prefer positive over neutral feedback?	12	2	0
Do you like negative feedback?	6	7	1

to be polite. It didn't feel to be rude as well." Similarly, MP14 thought PairBuddy's tone achieved a good balance, saying, *"And the tone was like, not very formal, which I liked, but I wouldn't want it more casual than it is now."*

Human vs. robotic style: Participants had mixed opinions on whether PairBuddy should be human-like or robotic. 5/14 participants favored human-like dialogue, while 3/14 wished PairBuddy's dialogue was more robotic. MS3 idealized a more human PairBuddy, explaining, *"Ultimately, I think having it speak as human-like as possible is the goal."* MP10 described that he enjoyed the social aspect of PairBuddy's human-like dialogue, *"I enjoy... my buddies and working with them... Closer to that, I think more people would like [PairBuddy] because that is how they work in real world."* Participants who preferred robotic dialogue mentioned that it would be strange or disturbing if PairBuddy was more human-like, including MS1, who commented, *"It would be weirder if it [PairBuddy] was more like a human; You kind of expect some level of not human dialogue."* Similarly, MS5 explained her feelings toward PairBuddy's style, *"I think there's something a little creepy about something that's like really close to being human, but not quite."*

Positive/Neutral/Negative Feedback: All participants preferred (12/14) or were neutral (2/14) toward PairBuddy's positive feedback. MS4 voiced her opinion, saying, *"It [PairBuddy] wasn't rude about it. He was very like, 'I don't know if that's gonna work,' rather than like, 'Wow, why'd you do that?' I think it made it... a little more trustworthy."* MP10 emphasized the need for exclusively positive feedback, *"The way [humans] think and the way they solve problems requires a lot of motivation and requires a lot of positive praise, even if they are doing things wrong."*

Participants interpreted negative feedback as identifying mistakes in the code, so most of them liked (6/14) or were neutral (7/14) towards such responses. This misunderstanding was evidenced by MS8's comment, *"If there are negative parts to what I'm doing, I'd like to know."*

`Future Design`: We will provide personalization for PairBuddy's script across tone, style, and feedback (details in Section 5). To avoid the "uncanny valley" effect where realism becomes unsettling [70], PairBuddy's movement and stylization will be further considered. Finally, PairBuddy will continue to give encouragement while still providing informative feedback.

(3) Pair Jelling with PairBuddy as a Programming Partner

Participants' interactions with PairBuddy were fewer in the first half of sessions, but increased by the second half. Infrequent interactions may have been caused by participants' unfamiliarity with PairBuddy as evidenced by MS7's interview, *"It was a little bit awkward initially to figure out what things I could say that it would actually respond to... But I figured that out pretty quickly once we started working more."* Adjusting to PairBuddy in this way is similar to the pair "jelling" period of human-human pair programming, where programmers take time to get accustomed to each other's personality, style, and abilities [94]. However, the jelling period with PairBuddy often ended when participants understood its utility as MS5 explained, *"Seeing consistent results in regards to... positively affecting my performance would increase my trust in it [PairBuddy]."* Research corroborates this claim, as potential productivity has shown to be a significant motivator for the utilization of conversation agents [27].

(4) Unrealistic Assumptions Regarding PairBuddy's Capabilities

Our participants assumed that PairBuddy would know all the answers as described by PS1, *"I assumed it [PairBuddy] knew the answer."* Similarly, PS11 commented, *"I went in assuming that... you guys had already tested it quite a bit, and so the robot was really familiar with what the program should look like."* This assumption of PairBuddy's knowledge is untrue since design decisions `T1` and `T2` provide PairBuddy with only limited abilities since automatic code and test case generation are topics of ongoing research. In the future, the inaccurate expectations of PairBuddy's abilities can be addressed through further clarifications of PairBuddy's limitations in its script, *"Sorry, I have limited capabilities. Researchers are working to make me smart enough to answer all your questions. Follow the link I sent in the text chat to read the current research on automated code and test case generation."* However, PS3 understood PairBuddy's limitations, especially for more complicated tasks that require more discussions, *"I guess it depends on the complexity and the nuance of the situation and the [user] stories that I need to take care of, so for simpler problems or more straightforward tasks, I'd say that prefer PairBuddy. But for more complex scenarios, I would rather have a human that can provide more... pointed and specific feedback and advice."*

(5) Helpful as a Non-Judgemental Partner and When Working Solo

PairBuddy helped participants by acting as a non-judgemental partner. PS5 explained, *"I think the big thing is sometimes as a programmer, it's embarrassing when you make mistakes, so you're stuck on something around your colleagues, but PairBuddy I don't think judges me."* PS2 described the lack of pressure to perform, *"I feel like whenever I program alone, there's less pressure to like write something that's correct the first time... So I feel like in the end, it [PairBuddy] helped me write better code."*

5 DISCUSSIONS AND FUTURE WORK

Our results indicate that PairBuddy was an effective pair programming partner and was enjoyed by our study participants. For the future, the design space as well as the associated challenges are as follows:

(1) Supporting Method-level Code and Test Case Generation

Designing a conversational agent for the programming domain is challenging. Unlike other conversational agents, PairBuddy must uniquely support all stages of software development (i.e., clarifying requirements, discussing ideas, designing solutions, and implementing code). Support for each phase requires different knowledge and a different approach.

Moving forward, we need to research automated test case, code, and explainable feedback generation for PairBuddy. Currently, the language models such as GPT-3 [29] can be trained from Github [29, 77–79] to generate code, while automated test cases can be generated by tools such as Randoop [141] and EvoSuite [65, 66, 154]. However, these tools must be adapted in order for PairBuddy to explain the decisions made and give appropriate feedback. To directly answer the "why" questions that programmers ask, Ko et al. [105, 106] created the automated tool Whyline for both Alice [42] and Java, but similar research must be conducted in other programming languages and domains to allow agents to more directly support the refactoring stage of software development.

Furthermore, dialogues are often multimodal and involve both verbal and nonverbal inputs [23, 48, 52, 210]. Therefore, in the future, each dialogue template of PairBuddy's script will be accompanied by non-verbal meta-data (e.g., avatar facial expressions, UI events) to facilitate multimodal communication.

(2) Supporting Diverse Problem Solving Strategies

Additional problem-solving strategies must be integrated into PairBuddy to fully support all stages of the creative problem solving process. Problem-solving strategies such as backwards, divide and conquer, analogy, generalization, and "sleep on it" [112, 144, 198] enable programmers to make progress on their tasks. Previous research from Idea Garden [92, 93] shows that these problem-solving strategies can be implemented by presenting suggestions via language-independent templates, which are informed by language-dependent information about user tasks and progress. Based on Idea Garden, we integrated a backward problem-solving strategy in PairBuddy's script to support divergent thinking, but the future design will integrate additional problem-solving strategies inspired by Idea Garden.

(3) Supporting Learning Preferences: Solo or Collaboration

Research has shown that collaborative learning is more effective than traditional methods (such as lectures) since collaboration allows students to build their own mental models based on the discussion and knowledge transfer that occurs during the problem-solving process [11]. But still, many of our participants preferred to work solo. In the pilot study, some participants only used PairBuddy as a resource for code examples, while in the main study, a few preferred that PairBuddy only observe and identify errors. Still, others may prefer full collaboration to enjoy the benefits of pair programming. To support the diversity of learning preferences (solo vs. collaborative) among programmers, we will provide different options for PairBuddy's behavior including "Navigator Only," "Driver Only," or "Full Pair Programmer" modes.

(4) Implementing PairBuddy as a Task-Oriented vs. Non-Task-Oriented Agent

PairBuddy's design needs to fall between the two common types of conversational agents: task-oriented and non-task-oriented. Task-oriented agents perform a variety of tasks and services for users (e.g., Amazon's Alexa [188]) and provide concise, direct answers to user queries based on knowledge sources (e.g., Bing QA [196]). Non-task-oriented agents facilitate natural interaction between humans and electronic devices (e.g., Meena [8]). For pair programming, a task-oriented agent can assist user queries, while a non-task-oriented agent can engage in rapport making and off-topic dialogue. Research [39] has even identified methods to integrate both task- and non-task-oriented agents together. In our human-agent study, we found that 89.28% (3967/4443) of programmers' utterances were task-oriented. The abundance of task-oriented dialogue suggests that it may be advantageous for the future to implement PairBuddy as a task-oriented conversational agent that utilizes a pipeline architecture for intent classification, state tracking, and response generation. Subsequently, we must collect a large enough dataset of pair programming dialogue to train machine learning models to generate automated responses to programmers' queries.

Furthermore, we must consider the choice of datasets used for both task- and non-task-oriented conversational agents, since differences exist between human-human and human-agent dialogues. In human-human dialogue, changing roles is not often explicit [151], whereas in human-agent dialogue, role exchanges involve verbal permission and acceptance. Additionally, lengthy discussions about ideas are more prevalent in human-human than in human-agent conversations. The difference in dialogue style may require that the natural language processing, dialogue state tracking, and dialogue policy components of future conversational agents consider training their machine learning models on human-agent conversations rather than human-human.

(5) Designing for Varied Expertise of Programmers

Our main study explored the similarities and differences between expert (professional) and novice (student) programmers. However, self-efficacy scores and interview questions revealed only marginal differences. For instance, students considered PairBuddy's feelings (5/8) much more than professionals (1/6), suggesting that professionals used PairBuddy more as a tool rather than a partner. However, participants' familiarity in the applicable domain (Java) may have mattered more, as two professionals (MP13 and MP14) had less experience than many students. Particularly, our most experienced professional (MP10) uniquely ignored PairBuddy, opposed negative feedback, and excelled at the task. In regard to trusting PairBuddy, there was a slight discrepancy: 3/6 professionals vs. 5/8 students. While our interviews weren't conclusive, individuals with particularly low or high experience with a domain tend to trust computer answers more [37, 206]. Hence, future work will focus more specifically on the range of programming experience within both educational and professional settings.

(6) Manifesting Anthropomorphic Features of a Programmer

Although we strive to integrate anthropomorphic features into PairBuddy, we received mixed reactions from our participants. While PairBuddy's voice felt natural to participants, its avatar was less impactful, and many participants even minimized the avatar window. Preferences toward a more robotic or a more human-like PairBuddy were very mixed as well. For the past 20 years, researchers have argued in favor or against including anthropomorphic features in intelligent agents [72, 103]. Therefore, it is still an open-ended question whether PairBuddy should use features such as embodiment or emotional intelligence. In future studies, we will investigate whether anthropomorphic features are appropriate for PairBuddy.

Preferences toward PairBuddy's dialogue and interactions varied across many dimensions (e.g., human-like vs. robotic, positive vs. negative feedback). Participants' opinions often contradict one another, hindering the design of a universally accessible PairBuddy. Even for broadly accepted preferences (e.g., casual vs. polite), opinions will inevitably diverge with a larger sample size. To accommodate individual differences, PairBuddy's future design should be malleable, allowing users to tune PairBuddy's parameters to their liking [15]. One possible solution is to make PairBuddy's script interchangeable. For example, a robotic version of PairBuddy's script might say, *"I'm 80% certain that there is an error on line 45,"* while a casual version would say, *"I think we might have made a mistake on line 45."* However, interchangeable scripts only go so far, so in the future, we will explore additional avenues to provide a personalized PairBuddy experience.

6 CONCLUSION

In this research, we explore the uncharted territory of interactive pair programming conversational agents through the user-centered prototyping of our agent – PairBuddy. This work makes several contributions, including ones that generalize beyond pair programming:

- Anthropomorphic properties of PairBuddy arose from integrating diverse interface and interaction mechanisms (embodiment, dialogue styles, and agent actions) and programmer characteristics (technical skills and soft skills). These properties stem from an integration of novel concepts from our extensive literature review of various domains such as human-computer interaction, software engineering, artificial intelligence, psychology, and education. These properties can be utilized for advancing programmers' interactions in intelligent-tutoring systems and interactive educational platforms.
- Our focused design, evaluation, and refinement cycles through the use of two Wizard of Oz studies incrementally evolved PairBuddy's functionality to realize a robust conversational agent for pair programming. This approach can drive the design for programming conversational agents in other domains of programming including educating children, end-user programmers, and people with disabilities.
- The wizard's script and study materials for both pilot [5] and main studies [4] are available online for reproducibility by researchers and practitioners.

Our study results showed programmers' positive attitudes towards using PairBuddy. The results confirm the feasibility of PairBuddy as a programming partner that can significantly advance programmer-computer interactions. PairBuddy has significant potential to change how programming is learned and how programming is done. In the words of one of our participants with 20+ years of experience, *"I think what I learned out of this [study] is that [PairBuddy has] a lot of potential and... it excites me a lot where technology is going."*

ACKNOWLEDGMENTS

We would like to thank David Magar, Jarow Myers, Sam Gurka, Katherine Kwasny, and Bali Ong for help with the studies; David Piorkowski and Rachel Bellamy for revising; and Courtney Spivey for editing.

REFERENCES

[1] 2017. ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. *ISO/IEC/IEEE 24765:2017(E)* (2017), 1–541. https://doi.org/10.1109/IEEESTD.2017.8016712
[2] 2018. ISO/IEC/IEEE International Standard - Systems and software engineering — Developing information for users in an agile environment. *ISO/IEC/IEEE 26515:2018(E)* (2018), 1–32. https://doi.org/10.1109/IEEESTD.2018.8584455
[3] 2020. GitHub. http://github.com
[4] 2020. Main Study Supporting Material. https://drive.google.com/drive/folders/179OnUEVqPHQjW_K38ynUopyL4MDKlG-u?usp=sharing
[5] 2020. Pilot Study Supporting Material. https://drive.google.com/drive/folders/1WgINmA_iz3iONfpbAfNB70oCN25mJ76k?usp=sharing
[6] 2020. StackOverflow. http://stackoverflow.com
[7] Gregory D. Abowd and Alan J. Dix. 1992. Giving undo attention. *Interacting with Computers* 4, 3 (12 1992), 317–342. https://doi.org/10.1016/0953-5438(92)90021-7 arXiv:https://academic.oup.com/iwc/article-pdf/4/3/317/2175174/iwc4-0317.pdf
[8] Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan, Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. 2020. Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977* (2020).
[9] Ali Ahmadvand, Jason Ingyu Choi, and Eugene Agichtein. 2019. Contextual dialogue act classification for open-domain conversational agents. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1273–1276.

[10] B. Al-Ani and D. Redmiles. 2009. In Strangers We Trust? Findings of an Empirical Study of Distributed Teams. In *2009 Fourth IEEE International Conference on Global Software Engineering*. 121–130.

[11] Maryam Alavi. 1994. Computer-mediated collaborative learning: An empirical evaluation. *MIS quarterly* (1994), 159–174.

[12] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege. 2010. A Systematic Review of the Application and Empirical Investigation of Search-Based Test Case Generation. *IEEE Transactions on Software Engineering* 36, 6 (2010), 742–762.

[13] Teresa M. Amabile and Michael G. Pratt. 2016. The dynamic componential model of creativity and innovation in organizations: Making progress, making meaning. *Research in Organizational Behavior* 36 (2016), 157 – 183. https://doi.org/10.1016/j.riob.2016.10.001

[14] Saleema Amershi, Dan Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. ACM, New York, NY, USA, Article 3, 13 pages. https://doi.org/10.1145/3290605.3300233

[15] Ofer Arazy, Oded Nov, and Nanda Kumar. 2015. Personalityzation: UI personalization, theoretical grounding in HCI and design research. *AIS Transactions on Human-Computer Interaction* 7, 2 (2015), 43–69.

[16] Michael Armstrong. 2012. *Armstrong's handbook of reward management practice: Improving performance through reward* (12 ed.). Kogan Page Publishers.

[17] Zahra Ashktorab, Mohit Jain, Q. Vera Liao, and Justin D. Weisz. 2019. Resilient Chatbots: Repair Strategy Preferences for Conversational Breakdowns. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article 254, 12 pages. https://doi.org/10.1145/3290605.3300484

[18] Claudio Barra and Broderick Crawford. 2007. Fostering Creativity Thinking in Agile Software Development, Vol. 4799. 415–426. https://doi.org/10.1007/978-3-540-76805-0_37

[19] Amy L. Baylor and Soyoung Kim. 2009. Designing nonverbal communication for pedagogical agents: When less is more. *Computers in Human Behavior* 25, 2 (2009), 450 – 457. https://doi.org/10.1016/j.chb.2008.10.008 Including the Special Issue: State of the Art Research into Cognitive Load Theory.

[20] A. Belshee. 2005. Promiscuous pairing and beginner's mind: Embrace inexperience. 125 – 131. https://doi.org/10.1109/ADC.2005.37

[21] Gary Bente, Sabine Rüggenberg, Nicole C. Krämer, and Felix Eschenburg. 2008. Avatar-Mediated Networking: Increasing Social Presence and Interpersonal Trust in Net-Based Collaborations. *Human Communication Research* 34, 2 (04 2008), 287–318. https://doi.org/10.1111/j.1468-2958.2008.00322.x arXiv:https://academic.oup.com/hcr/article-pdf/34/2/287/22325251/jhumcom0287.pdf

[22] Timothy Bickmore and Justine Cassell. 2001. Relational agents: a model and implementation of building user trust. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 396–403.

[23] Dan Bohus, Chit W Saw, and Eric Horvitz. 2014. Directions robot: in-the-wild experiences and lessons learned. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. 637–644.

[24] Yvonne Brackbill, William E Boblitt, Douglas Davlin, and John E Wagner. 1963. Amplitude of response and the delay-retention effect. *Journal of Experimental Psychology* 66, 1 (1963), 57.

[25] Jay Bradley, David Benyon, Oli Mival, and Nick Webb. 2010. Wizard of Oz experiments and companion dialogues. In *Proceedings of the 24th BCS Interaction Specialist Group Conference*. British Computer Society, 117–123.

[26] Sheryl Brahnam and Antonella De Angeli. 2012. Gender affordances of conversational agents. *Interacting with Computers* 24, 3 (04 2012), 139–153. https://doi.org/10.1016/j.intcom.2012.05.001 arXiv:https://academic.oup.com/iwc/article-pdf/24/3/139/2027399/iwc24-0139.pdf

[27] Petter Bae Brandtzaeg and Asbjørn Følstad. 2017. Why people use chatbots. In *International Conference on Internet Science*. Springer, 377–392.

[28] Tim Brown. 2009. *Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation*. HarperBusiness.

[29] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[30] Margaret Burnett, Anicia Peters, Charles Hill, and Noha Elarief. 2016. Finding Gender-Inclusiveness Software Issues with GenderMag: A Field Investigation. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2586–2598.

[31] Ramón Burri. 2018. *Improving user trust towards conversational chatbot interfaces with voice output.* Master's thesis. KTH, School of Electrical Engineering and Computer Science (EECS).

[32] Justine Cassell, Yukiko I Nakano, Timothy W Bickmore, Candace L Sidner, and Charles Rich. 2001. Non-verbal cues for discourse structure. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. 114–123.

[33] Mehmet Celepkolu and Kristy Elizabeth Boyer. 2018. Thematic Analysis of Students' Reflections on Pair Programming in CS1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. ACM, New York, NY, USA, 771–776. https://doi.org/10.1145/3159450.3159516

[34] Christopher P Cerasoli, Jessica M Nicklin, and Michael T Ford. 2014. Intrinsic motivation and extrinsic incentives jointly predict performance: A 40-year meta-analysis. *Psychological bulletin* 140, 4 (2014), 980.

[35] Hyun Jin Cha, Yong Se Kim, Seon Hee Park, Tae Bok Yoon, Young Mo Jung, and Jee-Hyong Lee. 2006. Learning styles diagnosis based on user interface behaviors for the customization of learning interfaces in an intelligent tutoring system. In *International Conference on Intelligent Tutoring Systems*. Springer, 513–524.

[36] Gary Charness and Uri Gneezy. 2012. Strong Evidence for Gender Differences in Risk Taking. *Journal of Economic Behavior & Organization* 83, 1 (2012), 50–58.

[37] J. Y. C. Chen and M. J. Barnes. 2014. Human–Agent Teaming for Multirobot Control: A Review of Human Factors Issues. *IEEE Transactions on Human-Machine Systems* 44, 1 (2014), 13–29.

[38] K. S. Choi. 2013. Evaluating Gender Significance within a Pair Programming Context. In *2013 46th Hawaii International Conference on System Sciences*. 4817–4825.

[39] Tai-Liang Chou and Yu-Ling Hsueh. 2019. A Task-Oriented Chatbot Based on LSTM and Reinforcement Learning. In *Proceedings of the 2019 3rd International Conference on Natural Language Processing and Information Retrieval* (Tokushima, Japan) *(NLPIR 2019)*. Association for Computing Machinery, New York, NY, USA, 87–91. https://doi.org/10.1145/3342827.3342844

[40] Alistair Cockburn and Laurie Williams. 2001. Extreme Programming Examined. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, Chapter The Costs and Benefits of Pair Programming, 223–243. http://dl.acm.org/citation.cfm?id=377517.377531

[41] Michelle Cohn, Chun-Yen Chen, and Zhou Yu. 2019. A Large-Scale User Study of an Alexa Prize Chatbot: Effect of TTS Dynamism on Perceived Quality of Social Dialog. In *Proceedings of the 20th Annual SIGdial Meeting on Discourse and Dialogue*. 293–306.

[42] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. *J. Comput. Sci. Coll.* 15, 5 (April 2000), 107–116.

[43] Tyne Crow, Andrew Luxton-Reilly, and Burkhard Wünsche. 2018. Intelligent tutoring systems for programming education: a systematic review. 53–62. https://doi.org/10.1145/3160489.3160492

[44] Fabio QB da Silva, Catarina Costa, A Cesar C Franca, and Rafael Prikladinicki. 2010. Challenges and solutions in distributed software development project management: A systematic literature review. In *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on*. IEEE, 87–96.

[45] Nils Dahlbäck, Arne Jönsson, and Lars Ahrenberg. 1993. Wizard of Oz studies: why and how. In *Proceedings of the 1st international conference on Intelligent user interfaces*. 193–200.

[46] M. Day, M. R. Penumala, and J. Gonzalez-Sanchez. 2019. Annete: An Intelligent Tutoring Companion Embedded into the Eclipse IDE. In *2019 IEEE First International Conference on Cognitive Machine Intelligence (CogMI)*. 71–80.

[47] Claudio León de la Barra and Broderick Crawford. 2007. Fostering Creativity Thinking in Agile Software Development. In *HCI and Usability for Medicine and Health Care*, Andreas Holzinger (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 415–426.

[48] Harm De Vries, Florian Strub, Sarath Chandar, Olivier Pietquin, Hugo Larochelle, and Aaron Courville. 2017. Guess-what?! visual object discovery through multi-modal dialogue. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5503–5512.

[49] Edward L Deci, Anja H Olafsen, and Richard M Ryan. 2017. Self-determination theory in work organizations: The state of a science. *Annual Review of Organizational Psychology and Organizational Behavior* 4 (2017), 19–43.

[50] Doris M. Dehn and Susanne van Mulken. 2000. The Impact of Animated Interface Agents: A Review of Empirical Research. *Int. J. Hum.-Comput. Stud.* 52, 1 (Jan. 2000), 1–22. https://doi.org/10.1006/ijhc.1999.0325

[51] Tom DeMarco and Timothy Lister. 1987. *Peopleware: Productive Projects and Teams.* Dorset House Publishing Co., Inc., New York, NY, USA.

[52] David DeVault, Ron Artstein, Grace Benn, Teresa Dey, Ed Fast, Alesia Gainer, Kallirroi Georgila, Jon Gratch, Arno Hartholt, Margaux Lhommet, et al. 2014. SimSensei Kiosk: A virtual human interviewer for healthcare decision support. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. 1061–1068.

[53] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[54] Eclipse 2019. Eclipse IDE. https://www.eclipse.org/

[55] Eclipse 2020. Junit. https://junit.org/junit5/

[56] Eclipse Plugin [n.d.]. An Eclipse Plug-in for Distributed Party Programming. www.saros-project.org
[57] Berland Edelman and Inc. 2010. *Creativity and education: Why it matters.* Retrieved September 18th, 2019 from http://www.adobe.com/aboutadobe/pressroom/pdfs/Adobe_Creativity_and_Education_Why_It_Matters_study.pdf
[58] R. Elghondakly, S. Moussa, and N. Badr. 2015. Waterfall and agile requirements-based model for automated test cases generation. In *2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS).* 607–612.
[59] Martha Evens and Joel Michael. 2006. *One-on-one tutoring by humans and computers.* Psychology Press.
[60] FACSvatar 2018. FACSvatar. https://github.com/NumesSanguis/FACSvatar
[61] Raoul Festante. 2007. *An introduction to the theory of gender-neutral language.* BoD–Books on Demand.
[62] FFmpeg 2016. FFmpeg. http://ffmpeg.org/.
[63] Carmen Fischer, Charlotte P. Malycha, and Ernestine Schafmann. 2019. The Influence of Intrinsic Motivation and Synergistic Extrinsic Motivators on Creativity and Innovation. *Frontiers in Psychology* 10 (2019), 137. https://doi.org/10.3389/fpsyg.2019.00137
[64] Cyrus K. Foroughi, Nicole E. Werner, Erik T. Nelson, and Deborah A. Boehm-Davis. 2014. Do Interruptions Affect Quality of Work? *Human Factors* 56, 7 (2014), 1262–1271. https://doi.org/10.1177/0018720814531786 arXiv:https://doi.org/10.1177/0018720814531786 PMID: 25490806.
[65] Gordon Fraser and Andrea Arcuri. 2012. Whole test suite generation. *IEEE Transactions on Software Engineering* 39, 2 (2012), 276–291.
[66] Gordon Fraser and Andrea Arcuri. 2013. Evosuite: On the challenges of test case generation in the real world. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation.* IEEE, 362–369.
[67] Freelancing Platform [n.d.]. Upwork Inc.
[68] Hans Gallis and Erik Arisholm. 2002. A transition from partner programming to pair programming-an Industrial Case Study.
[69] Jianfeng Gao, Michel Galley, and Lihong Li. 2018. Neural approaches to conversational ai. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval.* 1371–1374.
[70] Tom Geller. 2008. Overcoming the uncanny valley. *IEEE computer graphics and applications* 28, 4 (2008), 11–17.
[71] GenderMag 2019. GenderMag. http://gendermag.org/
[72] Stella George. 2019. From Sex and Therapy Bots to Virtual Assistants and Tutors: How Emotional Should Artificially Intelligent Agents Be?. In *Proceedings of the 1st International Conference on Conversational User Interfaces* (Dublin, Ireland) *(CUI '19)*. Association for Computing Machinery, New York, NY, USA, Article 19, 3 pages. https://doi.org/10.1145/3342775.3342807
[73] Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and L. Thomas van Binsbergen. 2016. Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback. *International Journal of Artificial Intelligence in Education* 27 (02 2016). https://doi.org/10.1007/s40593-015-0080-x
[74] Barney G. Glaser and Anselm L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine de Gruyter, New York, NY.
[75] M. Gonzalez-Franco, A. Steed, S. Hoogendyk, and E. Ofek. 2020. Using Facial Animation to Increase the Enfacement Illusion and Avatar Self-Identification. *IEEE Transactions on Visualization and Computer Graphics* 26, 5 (2020), 2023–2029.
[76] Paul Green and Lisa Wei-Haas. 1985. *The Wizard of Oz: a tool for rapid development of user interfaces.*
[77] GTP-3 [n.d.]. Tweet of code generated from GPT-3. https://twitter.com/sharifshameem/status/1282676454690451457
[78] GTP-3 [n.d.]. YouTube video showing generation of code by GPT3. https://www.youtube.com/watch?v=utuz7wBGjKM
[79] GTP-3 [n.d.]. YouTube video showing generation of code by GPT3. GPT3:https://www.youtube.com/watch?v=y5-wzgIySb4
[80] GTTS 2019. Google text-to-speech python library. https://github.com/pndurette/gTTS
[81] Qinghong Han, Yuxian Meng, Fei Wu, and Jiwei Li. 2020. Non-Autoregressive Neural Dialogue Generation. *arXiv preprint arXiv:2002.04250* (2020).
[82] Brian F. Hanks. 2004. Distributed Pair Programming: An Empirical Study. In *Extreme Programming and Agile Methods - XP/Agile Universe 2004*, Carmen Zannier, Hakan Erdogmus, and Lowell Lindstrom (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 81–91.
[83] Dai Hasegawa, Justine Cassell, and Kenji Araki. 2010. The Role of Embodiment and Perspective in Direction-Giving Systems. (01 2010).
[84] Khaled Hassanein and Milena Head. 2007. Manipulating perceived social presence through the web interface and its impact on attitude towards online shopping. *International Journal of Human-Computer Studies* 65, 8 (2007), 689 – 708. https://doi.org/10.1016/j.ijhcs.2006.11.018
[85] John Hattie. 1999. Influences on student learning. *Inaugural lecture given on August* 2, 1999 (1999), 21.

[86] John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research* 77, 1 (2007), 81–112.

[87] Renate Häuslschmid, Max von Bülow, Bastian Pfleging, and Andreas Butz. 2017. SupportingTrust in Autonomous Driving. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces* (Limassol, Cyprus) *(IUI '17)*. ACM, New York, NY, USA, 319–329. https://doi.org/10.1145/3025171.3025198

[88] Xinting Huang, Jianzhong Qi, Yu Sun, and Rui Zhang. 2020. Semi-Supervised Dialogue Policy Learning via Stochastic Reward Estimation. *arXiv preprint arXiv:2005.04379* (2020).

[89] Scott G. Isaksen and Donald J. Treffinger. 2004. Celebrating 50 years of Reflective Practice: Versions of Creative Problem Solving. *The Journal of Creative Behavior* 38, 2 (June 2004), 75–101.

[90] Mohit Jain, Pratyush Kumar, Ishita Bhansali, Q. Vera Liao, Khai Truong, and Shwetak Patel. 2018. FarmChat: A Conversational Agent to Answer Farmer Queries. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2, 4, Article 170 (Dec. 2018), 22 pages. https://doi.org/10.1145/3287048

[91] Mohit Jain, Pratyush Kumar, Ramachandra Kota, and Shwetak N Patel. 2018. Evaluating and informing the design of chatbots. In *Proceedings of the 2018 Designing Interactive Systems Conference*. 895–906.

[92] William Jernigan, Amber Horvath, Michael Lee, Margaret Burnett, Taylor Cuilty, Sandeep Kuttal, Anicia Peters, Irwin Kwan, Faezeh Bahmani, Andrew Ko, and Christopher Mendez. 2017. General Principles for a Generalized Idea Garden. *Journal of Visual Languages & Computing* (05 2017). https://doi.org/10.1016/j.jvlc.2017.04.005

[93] Will Jernigan, Amber Horvath, Michael Lee, Margaret Burnett, Cuilty Taylor, Sandeep Kuttal, Anicia Peters, Irwin Kwan, Faezeh Bahmani, and Andrew Ko. 2015. A Principled Evaluation for a Principled Idea Garden. https://doi.org/10.1109/VLHCC.2015.7357222

[94] Danielle Jones and Scott Fleming. 2013. What use is a backseat driver? A qualitative investigation of pair programming. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, 103–110. https://doi.org/10.1109/VLHCC.2013.6645252

[95] Ankur Joshi, Saket Kale, Satish Chandel, and D Kumar Pal. 2015. Likert scale: Explored and explained. *Current Journal of Applied Science and Technology* (2015), 396–403.

[96] Ewa Kacewicz, James W. Pennebaker, Matthew Davis, Moongee Jeon, and Arthur C. Graesser. 2014. Pronoun Use Reflects Standings in Social Hierarchies. *Journal of Language and Social Psychology* 33, 2 (2014), 125–143. https://doi.org/10.1177/0261927X13502654 arXiv:https://doi.org/10.1177/0261927X13502654

[97] Peter H. Kahn, Nathan G. Freier, Takayuki Kanda, Hiroshi Ishiguro, Jolina H. Ruckert, Rachel L. Severson, and Shaun K. Kane. 2008. Design Patterns for Sociality in Human-Robot Interaction. In *Proceedings of the 3rd ACM/IEEE International Conference on Human Robot Interaction* (Amsterdam, The Netherlands) *(HRI '08)*. Association for Computing Machinery, New York, NY, USA, 97–104. https://doi.org/10.1145/1349822.1349836

[98] Neha Katira, Laurie Williams, Laurie Williams, Eric Wiebe, Carol Miller, Suzanne Balik, and Ed Gehringer. 2004. On Understanding Compatibility of Student Pair Programmers. *SIGCSE Bull.* 36, 1 (March 2004), 7–11. https://doi.org/10.1145/1028174.971307

[99] R. K. Kavitha and M. S. Irfan Ahmed. 2013. Knowledge Sharing Through Pair Programming in Learning Environments: An empirical study. *Education and Information Technologies* 20, 2 (Oct. 2013), 319–333.

[100] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting Working Code Examples. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 664–675. https://doi.org/10.1145/2568225.2568292

[101] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. 2018. FaCoY: A Code-to-Code Search Engine. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 946–957. https://doi.org/10.1145/3180155.3180187

[102] Sungdong Kim, Sohee Yang, Gyuwan Kim, and Sang-Woo Lee. 2019. Efficient Dialogue State Tracking by Selectively Overwriting Memory. *arXiv preprint arXiv:1911.03906* (2019).

[103] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. 2017. The Rise of Bots: A Survey of Conversational Interfaces, Patterns, and Paradigms. In *Proceedings of the 2017 Conference on Designing Interactive Systems* (Edinburgh, United Kingdom) *(DIS '17)*. Association for Computing Machinery, New York, NY, USA, 555–565. https://doi.org/10.1145/3064663.3064672

[104] Avraham N Kluger and Angelo DeNisi. 1996. The effects of feedback interventions on performance: A historical review, a meta-analysis, and a preliminary feedback intervention theory. *Psychological bulletin* 119, 2 (1996), 254.

[105] Andrew J. Ko and Brad A. Myers. 2004. Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria) *(CHI '04)*. ACM, New York, NY, USA, 151–158. https://doi.org/10.1145/985692.985712

[106] Andrew J. Ko and Brad A. Myers. 2009. Finding Causes of Program Output with the Java Whyline. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Boston, MA, USA) *(CHI '09)*. ACM, New York, NY, USA, 1569–1578. https://doi.org/10.1145/1518701.1518942

[107] James A. Kulik and Chen-Lin C. Kulik. 1988. Timing of Feedback and Verbal Learning. *Review of Educational Research* 58, 1 (1988), 79–97. https://doi.org/10.3102/00346543058001079 arXiv:https://doi.org/10.3102/00346543058001079

[108] Sandeep Kuttal, Kevin Gerstner, and Alexandra Bejarano. 2019. Remote Pair-Programming in Online CS Education: Investigating through a Gender Lens. *IEEE Symposium on Visual Languages & Human-Centric Computing* (2019).

[109] S. K. Kuttal, J. Myers, S. Gurka, D. Magar, D. Piorkowski, and R. Bellamy. 2020. Towards Designing Conversational Agents for Pair Programming: Accounting for Creativity Strategies and Conversational Styles. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–11.

[110] Thomas K Landauer. 1987. Psychology as a mother of invention. *ACM SIGCHI Bulletin* 18, 4 (1987), 333–335.

[111] Hung Le, Richard Socher, and Steven C.H. Hoi. 2020. Non-Autoregressive Dialog State Tracking. In *International Conference on Learning Representations*. https://openreview.net/forum?id=H1e_cC4twS

[112] Marvin Levine. 1988. *Effective problem solving*. Prentice Hall.

[113] Clayton Lewis. 1982. *Using the "thinking-aloud" method in cognitive interface design*. IBM T.J. Watson Research Center, Yorktown Heights, N.Y.

[114] Shaofeng Li. 2010. The effectiveness of corrective feedback in SLA: A meta-analysis. *Language learning* 60, 2 (2010), 309–365.

[115] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. 2007. Feature location via information retrieval based filtering of a single scenario execution trace. *ASE'07 - 2007 ACM/IEEE International Conference on Automated Software Engineering*, 234–243. https://doi.org/10.1145/1321631.1321667

[116] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 4487–4496.

[117] Zhiqiang Liu and Dieter J Schonwetter. 2004. Teaching creativity in engineering. *International Journal of Engineering Education* 20, 5 (2004), 801–808.

[118] Ju Long. 2009. Open Source Software Development Experiences on the Students' Resumes: Do They Count?-Insights from the Employers' Perspectives. *Journal of Information Technology Education: Research* 8, 1 (2009), 229–242.

[119] Irene Lopatovska and Harriet Williams. 2018. Personification of the Amazon Alexa: BFF or a Mindless Companion. In *Proceedings of the 2018 Conference on Human Information Interaction & Retrieval* (New Brunswick, NJ, USA) *(CHIIR '18)*. Association for Computing Machinery, New York, NY, USA, 265–268. https://doi.org/10.1145/3176349.3176868

[120] Ewa Luger and Abigail Sellen. 2016. " Like Having a Really Bad PA" The Gulf between User Expectation and Experience of Conversational Agents. In *Proceedings of the 2016 CHI conference on human factors in computing systems*. 5286–5297.

[121] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev. 2005. Static techniques for concept location in object-oriented code. In *13th International Workshop on Program Comprehension (IWPC'05)*. 33–42.

[122] Jennifer Marlow and Laura Dabbish. 2013. Activity traces and signals in software developer recruitment and hiring. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 145–156.

[123] Akane Matsushima, Natsuki Oka, Chie Fukada, and Kazuaki Tanaka. 2019. Understanding Dialogue Acts by Bayesian Inference and Reinforcement Learning. In *Proceedings of the 7th International Conference on Human-Agent Interaction* (Kyoto, Japan) *(HAI '19)*. Association for Computing Machinery, New York, NY, USA, 262–264. https://doi.org/10.1145/3349537.3352786

[124] Charlie McDowell, Linda Werner, Heather Bullock, and Julian Fernald. 2002. The Effects of Pair-programming on Performance in an Introductory Programming Course. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (Cincinnati, Kentucky) *(SIGCSE '02)*. ACM, New York, NY, USA, 38–42. https://doi.org/10.1145/563340.563353

[125] Charlie McDowell, Linda Werner, Heather E. Bullock, and Julian Fernald. 2003. The Impact of Pair Programming on Student Performance, Perception and Persistence. In *Proceedings of the 25th International Conference on Software Engineering* (Portland, Oregon) *(ICSE '03)*. IEEE Computer Society, Washington, DC, USA, 602–607. http://dl.acm.org/citation.cfm?id=776816.776899

[126] Phil McMinn. 2004. Search-based software test data generation: a survey: Research Articles. *Softw. Test., Verif. Reliab.* 14 (06 2004), 105–156. https://doi.org/10.1002/stvr.294

[127] Meiliana, Irwandhi Septian, Ricky Setiawan Alianto, Daniel, and Ford Lumban Gaol. 2017. Automated Test Case Generation from UML Activity Diagram and Sequence Diagram using Depth First Search Algorithm. *Procedia Computer Science* 116 (2017), 629 – 637. https://doi.org/10.1016/j.procs.2017.10.029 Discovery and innovation of computer science technology in artificial intelligence era: The 2nd International Conference on Computer Science and Computational Intelligence (ICCSCI 2017).

[128] Grigori Melnik and Frank Maurer. 2002. Perceptions of agile practices: A student survey. In *Conference on Extreme Programming and Agile Methods*. Springer, 241–250.

[129] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. 2018. Open Source Barriers to Entry, Revisited:

A Sociotechnical Perspective. In *Proceedings of the 40th International Conference on Software Engineering*. ACM, 1004–1015.

[130] Casey Miller and Kate Swift. 2001. *The handbook of nonsexist writing*. iUniverse.

[131] Matheus Monteiro, Erica Souza, Andre Endo, and Nandamudi Vijaykumar. 2019. Analyzing graph-based algorithms employed to generate test cases from finite state machines. https://doi.org/10.1109/LATW.2019.8704603

[132] Dana Movshovitz-Attias, Yair Movshovitz-Attias, Peter Steenkiste, and Christos Faloutsos. 2013. Analysis of the reputation system and user contributions on a question answering website: Stackoverflow. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ACM, 886–893.

[133] Susanne van Mulken, Elisabeth André, and Jochen Müller. 1999. An Empirical Study on the Trustworthiness of Life-like Interface Agents. In *Proceedings of the HCI International '99 (the 8th International Conference on Human-Computer Interaction) on Human-Computer Interaction: Communication, Cooperation, and Application Design-Volume 2 - Volume 2*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 152–156. http://dl.acm.org/citation.cfm?id=647944.741893

[134] Magdalene Ng, Kovila PL Coopamootoo, Ehsan Toreini, Mhairi Aitken, Karen Elliot, and Aad van Moorsel. 2020. Simulating the Effects of Social Presence on Trust, Privacy Concerns\& Usage Intentions in Automated Bots for Finance. *arXiv preprint arXiv:2006.15449* (2020).

[135] Jakob Nielsen and Rolf Molich. 1990. Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 249–256.

[136] Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Learning to rank code examples for code search engines. *Empirical Software Engineering* 22, 1 (2017), 259–291.

[137] John Noll, Sarah Beecham, and Ita Richardson. 2010. Global software development and collaboration: barriers and solutions. *ACM inroads* 1, 3 (2010), 66–78.

[138] Catherine S. Oh, Jeremy N. Bailenson, and Gregory F. Welch. 2018. A Systematic Review of Social Presence: Definition, Antecedents, and Implications. *Frontiers in Robotics and AI* 5 (2018), 114. https://doi.org/10.3389/frobt.2018.00114

[139] Andy Oram and Greg Wilson. 2010. *Making Software: What Really Works, and Why We Believe It* (1st ed.). O'Reilly Media, Inc.

[140] A.F. Osborn. 1957. *Applied Imagination: Principles and Procedures of Creative Thinking*. Charles Scribner's Sons.

[141] Carlos Pacheco and Michael D Ernst. 2007. Randoop: feedback-directed random testing for Java. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. 815–816.

[142] M. Page-Jones. 1988. *The Practical Guide to Structured Systems Design*. Prentice Hall.

[143] David Walsh Palmieri. 2002. Knowledge Management Through Pair Programming.

[144] George Polya. 2004. *How to solve it: A new aspect of mathematical method*. Vol. 85. Princeton university press.

[145] M. Raghothaman, Y. Wei, and Y. Hamadi. 2016. SWIM: Synthesizing What I Mean - Code Search and Idiomatic Snippet Synthesis. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 357–367.

[146] Vipul Raheja and Joel Tetreault. 2019. Dialogue Act Classification with Context-Aware Self-Attention. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 3727–3733.

[147] Prerana Pradeepkumar Rane. 2017. *Automatic generation of test cases for agile using natural language processing*. Ph.D. Dissertation. Virginia Tech.

[148] James L Reinertsen. 2000. Let's talk about error. *BMJ* 320, 7237 (2000), 730. https://doi.org/10.1136/bmj.320.7237.730 arXiv:https://www.bmj.com/content/320/7237/730.full.pdf

[149] Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

[150] Liliang Ren, Jianmo Ni, and Julian McAuley. 2019. Scalable and Accurate Dialogue State Tracking via Hierarchical Sequence Generation. *arXiv preprint arXiv:1909.00754* (2019).

[151] P. Robe, S. Kaur Kuttal, Y. Zhang, and R. Bellamy. 2020. Can Machine Learning Facilitate Remote Pair Programming? Challenges, Insights Implications. In *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–11.

[152] Fernando J. Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. 2017. Exploring the Pair Programming Process: Characteristics of Effective Collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. ACM, New York, NY, USA, 507–512. https://doi.org/10.1145/3017680.3017748

[153] Carl Ransom Rogers and Richard Evans Farson. 1957. *Active listening*. Industrial Relations Center of the University of Chicago Chicago, IL.

[154] José Miguel Rojas, José Campos, Mattia Vivanti, Gordon Fraser, and Andrea Arcuri. 2015. Combining multiple coverage criteria in search-based unit test generation. In *International Symposium on Search Based Software Engineering*. Springer, 93–108.

[155] Sherry Ruan, Jacob O. Wobbrock, Kenny Liou, Andrew Ng, and James A. Landay. 2018. Comparing Speech and Keyboard Text Entry for Short Messages in Two Languages on Touchscreen Phones. *Proc. ACM Interact. Mob. Wearable*

Ubiquitous Technol. 1, 4, Article 159 (Jan. 2018), 23 pages. https://doi.org/10.1145/3161187
[156] Omar Ruvalcaba, Linda Werner, and Jill Denner. 2016. Observations of Pair Programming: Variations in Collaboration Across Demographic Groups. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. ACM, New York, NY, USA, 90–95. https://doi.org/10.1145/2839509.2844558
[157] Anita Sarma, Xiaofan Chen, Sandeep Kuttal, Laura Dabbish, and Zhendong Wang. 2016. Hiring in the global stage: Profiles of online contributions. In *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on.* IEEE, 1–10.
[158] T. Savage, M. Revelle, and D. Poshyvanyk. 2010. FLAT3: feature location and textual tracing tool. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 2. 255–258.
[159] Marvin L Schroth and Elissa Lund. 1993. Role of delay of feedback on subsequent pattern recognition transfer tasks. *Contemporary Educational Psychology* 18, 1 (1993), 15–22.
[160] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.
[161] Michael Seymour, Kai Riemer, and Judy Kay. 2017. Interactive Realistic Digital Avatars-Revisiting the Uncanny Valley. (2017).
[162] Arun Shekhar and Nicola Marsden. 2018. Cognitive Walkthrough of a Learning Management System with Gendered Personas. In *Proceedings of the 4th Conference on Gender & IT*. ACM, 191–198.
[163] Ben Shneiderman. 1982. Designing computer system messages. *Commun. ACM* 25, 9 (1982), 610–611.
[164] Leif Singer, Fernando Figueira Filho, Brendan Cleary, Christoph Treude, Margaret-Anne Storey, and Kurt Schneider. 2013. Mutual assessment in the social programmer ecosystem: An empirical investigation of developer profile aggregators. In *Proceedings of the 2013 conference on Computer supported cooperative work*. ACM, 103–116.
[165] Social Bot [n.d.]. Chatbot Statistics. https://www.smallbizgenius.net/by-the-numbers/chatbot-statistics/#gref
[166] Social Bot [n.d.]. Cleverbot. https://www.cleverbot.com/
[167] Social Bot [n.d.]. Mitsuku. https://www.pandorabots.com/mitsuku/
[168] Social Bot [n.d.]. SAP Conversational AI. https://www.sap.com/products/conversational-ai.html
[169] Social Bot [n.d.]. Xiaoice AI Assistant. https://www.digitaltrends.com/cool-tech/xiaoice-microsoft-future-of-ai-assistants/
[170] Social Media Website [n.d.]. Discord. www.discord.com
[171] Social Media Website [n.d.]. Facebook. www.facebook.com
[172] Social Media Website [n.d.]. Google Hangouts. www.hangouts.google.com
[173] Social Media Website [n.d.]. Skype. www.skype.com
[174] Social Media Website [n.d.]. Twitter. www.twitter.com
[175] Software Application [n.d.]. Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS. https://www.electronjs.org/
[176] Software Application [n.d.]. Facerig. https://facerig.com/
[177] Software Application [n.d.]. VB-Audio VoiceMeeter. https://vb-audio.com/Voicemeeter/
[178] Jörg Spieler. [n.d.]. *UCDetector*. http://www.ucdetector.org/
[179] Lee Sproull, Mani Subramani, Sara Kiesler, Janet H. Walker, and Keith Waters. 1996. When the Interface is a Face. *Hum.-Comput. Interact.* 11, 2 (June 1996), 97–124. https://doi.org/10.1207/s15327051hci1102_1
[180] Saiying Steenbergen-Hu and Harris Cooper. 2014. A meta-analysis of the effectiveness of intelligent tutoring systems on college students' academic learning. *Journal of Educational Psychology* 106, 2 (2014), 331.
[181] Persis T Sturges. 1972. Information delay and retention: Effect of information in feedback and tests. *Journal of Educational Psychology* 63, 1 (1972), 32.
[182] Linda K Swindell and Walter F Walls. 1993. Response confidence and the delay retention effect. *Contemporary Educational Psychology* 18, 3 (1993), 363–375.
[183] Ryuichi Takanobu, Runze Liang, and Minlie Huang. 2020. Multi-agent task-oriented dialog policy learning with role-aware reward decomposition. *arXiv preprint arXiv:2004.03809* (2020).
[184] Diana-Cezara Toader, Grațiela Boca, Rita Toader, Mara Măcelaru, Cezar Toader, Diana Ighian, and Adrian T. Rădulescu. 2019. The Effect of Social Presence and Chatbot Errors on Trust. *Sustainability* 12, 1 (Dec 2019), 256. https://doi.org/10.3390/su12010256
[185] Susanne van Mulken, Elisabeth André, and Jochen Müller. 1998. The Persona Effect: How Substantial Is It?. In *People and Computers XIII*, Hilary Johnson, Lawrence Nigay, and Christopher Roast (Eds.). Springer London, London, 53–66.
[186] Kurt VanLehn. 2011. The Relative Effectiveness of Human Tutoring, Intelligent Tutoring Systems, and Other Tutoring Systems. *Educational Psychologist* 46, 4 (2011), 197–221. https://doi.org/10.1080/00461520.2011.611369
[187] Nicolas Vermeulen, Olivier Corneille, and Paula M. Niedenthal. 2008. Sensory load incurs conceptual processing costs. *Cognition* 109, 2 (2008), 287 – 294. https://doi.org/10.1016/j.cognition.2008.09.004
[188] Virtual Assistant [n.d.]. Amazon Alexa. https://developer.amazon.com/en-US/alexa

[189] Virtual Assistant [n.d.]. Apple Siri. https://www.apple.com/siri/
[190] Virtual Assistant [n.d.]. Google Assistant. https://assistant.google.com/
[191] Virtual Assistant [n.d.]. IBM Watson Assistant. https://cloud.ibm.com/apidocs/assistant/assistant-v2
[192] Virtual Assistant [n.d.]. Oracle Digital Assistant. https://www.oracle.com/solutions/chatbots/
[193] Aurora Vizcaíno. 2005. A Simulated Student Can Improve Collaborative Learning. *Int. J. Artif. Intell. Ed.* 15, 1 (Jan. 2005), 3–40.
[194] Tony Wagner and Robert A Compton. 2012. *Creating innovators: The making of young people who will change the world.* Simon and Schuster.
[195] Pierre Wargnier, Giovanni Carletti, Yann Laurent-Corniquet, Samuel Benveniste, Pierre Jouvelot, and Anne-Sophie Rigaud. 2016. Field evaluation with cognitively-impaired older adults of attention management in the embodied conversational agent louise. In *2016 IEEE International Conference on Serious Games and Applications for Health (SeGAH).* IEEE, 1–8.
[196] Website [n.d.]. Bing QA. https://www.microsoft.com/en-us/research/project/open-domain-question-answering/
[197] Linda L. Werner, Brian Hanks, and Charlie McDowell. 2004. Pair-programming Helps Female Computer Science Students. *J. Educ. Resour. Comput.* 4, 1, Article 4 (March 2004). https://doi.org/10.1145/1060071.1060075
[198] Wayne A Wickelgren. 1974. *How to solve problems: Elements of a theory of problems and problem solving.* WH Freeman San Francisco.
[199] Laurie Williams and Bob Kessler. 2000. The Effects of "Pair-Pressure" and "Pair-Learning" on Software Engineering Education. In *Proceedings of the 13th Conference on Software Engineering Education & Training (CSEET '00).* IEEE Computer Society, Washington, DC, USA, 59–. http://dl.acm.org/citation.cfm?id=794188.794326
[200] Laurie Williams and Robert Kessler. 2002. *Pair Programming Illuminated.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
[201] L. Williams, C. McDowell, N. Nagappan, J. Fernald, and L. Werner. 2003. Building pair programming knowledge through a family of experiments. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* 143–152. https://doi.org/10.1109/ISESE.2003.1237973
[202] Laurie A. Williams, Eric N. Wiebe, Kai Yang, Miriam Ferzli, and Carol Miller. 2002. In Support of Pair Programming in the Introductory Computer Science Course. *Computer Science Education* 12 (2002), 197–212.
[203] James Wilson and Daniel Rosenberg. 1988. Rapid prototyping for user interface design. In *Handbook of human-computer interaction.* Elsevier, 859–875.
[204] Ian H Witten, Craig G Nevill-Manning, and DL Maulsby. 1996. Interacting with learning agents: implications for ml from hci. In *Workshop on Machine Learning meets Human-Computer Interaction, ML*, Vol. 96. 51–58.
[205] Qingyang Wu, Yichi Zhang, Yu Li, and Zhou Yu. 2019. Alternating Recurrent Dialog Model with Large-scale Pre-trained Language Models. *arXiv preprint arXiv:1910.03756* (2019).
[206] Yvonne Wærn and Robert Ramberg. 1996. People's perception of human and computer advice. *Computers in Human Behavior* 12, 1 (1996), 17 – 27. https://doi.org/10.1016/0747-5632(95)00016-X
[207] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 5753–5763. http://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf
[208] Nick Yee, Jeremy N Bailenson, and Kathryn Rickertsen. 2007. A Meta-analysis of the Impact of the Inclusion and Realism of Human-like Faces on User Experiences in Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI '07).* ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/1240624.1240626
[209] Mohan Zalake, Julia Woodward, Amanpreet Kapoor, and Benjamin Lok. 2018. Assessing the Impact of Virtual Human's Appearance on Users' Trust Levels. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents* (Sydney, NSW, Australia) *(IVA '18).* Association for Computing Machinery, New York, NY, USA, 329–330. https://doi.org/10.1145/3267851.3267863
[210] Jiaping Zhang, Tiancheng Zhao, and Zhou Yu. 2018. Multimodal hierarchical reinforcement learning policy for task-oriented visual dialog. *arXiv preprint arXiv:1805.03257* (2018).
[211] Yichi Zhang, Zhijian Ou, and Zhou Yu. 2019. Task-Oriented Dialog Systems that Consider Multiple Appropriate Responses under the Same Context. *arXiv preprint arXiv:1911.10484* (2019).
[212] Yong Zhao. 2012. *World class learners: Educating creative and entrepreneurial students.* Corwin Press.
[213] R. Zhi, S. Marwan, Y. Dong, N. Lytle, T. W. Price, and T. Barnes. 2019. Toward Data-Driven Example Feedback for Novice Programming. In *Proceedings of the 12th International Conference on Educational Data Mining*.
[214] C. Zhou, S. K. Kuttal, and I. Ahmed. 2018. What Makes a Good Developer? An Empirical Study of Developers' Technical and Social Competencies. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing*

(VL/HCC). 319–321.

[215] Su Zhu, Jieyu Li, Lu Chen, and Kai Yu. 2020. Efficient Context and Schema Fusion Networks for Multi-Domain Dialogue State Tracking. *arXiv preprint arXiv:2004.03386* (2020).

[216] Franz Zieris and Lutz Prechelt. 2014. On Knowledge Transfer Skill in Pair Programming. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Torino, Italy) *(ESEM '14)*. ACM, New York, NY, USA, Article 11, 10 pages. https://doi.org/10.1145/2652524.2652529

7 AUTHOR STATEMENT

This work is not related to any prior or concurrent publications, and its contributions stand on its own.