Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Master Thesis

An LLM-Based Agent to Support Creative Problem-Solving during Programming Tasks

submitted by

Steffen Schubert

Matriculation Number 7194327

Course of Study: M.Sc. Computer Science

Faculty of Mathematics, Informatics and Natural Sciences

Department of Informatics

submitted on 19.05.2025

Supervisor: Christian Rahe M.Sc.

1st Examiner: Prof. Dr. Walid Maalej

2nd Examiner: Abir Bouraffa M.Sc.

Abstract

The rapid advancement of generative AI (GenAI) has significantly transformed software development by introducing powerful capabilities for AI-assisted programming. However, alongside this potential, concerns have been raised regarding the quality of generated code, impacts on metacognitive processes and the risk of overreliance on GenAI.

We introduce the *CPS Agent*, an LLM-based conversational agent guided by principles of the creative problem-solving process (CPS), featuring proactive behavior and aspects of emotional intelligence. This study investigates the effectiveness of the agent in supporting creative problem-solving, with particular emphasis on user perceptions of the agent as a pair programming partner and reliance on its support for problem-solving and code synthesis.

We conducted a comparative software development experiment involving a programming task to create a feature in a small, pre-existing codebase. Our findings indicate that the *CPS Agent* was perceived in more socially oriented roles compared to a baseline agent and offered more granular support across the phases of the creative problem-solving process. Furthermore, participants interacting with the *CPS Agent* exhibited reduced reliance on the agent for code synthesis.

The agent's emotional intelligence and proactive behavior received mixed responses. While some participants valued the collaborative interaction, others found it misaligned with their expectations of a utilitarian tool. Especially the proactive behavior was frequently perceived as disruptive and irrelevant.

Contents

1 Introduction

The development of capable generative artificial intelligence (GenAI) tools, such as *GitHub Copilot*[1] and *ChatGPT*[2], has significantly impacted the domain of knowledge work, such as software development [1, 2]. Through automatic code generation and problem-solving, such tools give rise to a huge potential for AI-assisted programming [3]. However, alongside this potential, concerns have been raised regarding the quality of generated code [4], impacts on metacognitive processes [5] and the risk of overreliance on GenAI [6].

One reason for these concerns is the capability of GenAI tools to provide readily available solutions on demand or even automatically, which may encourage users to adopt these suggestions without engaging in deeper reflection or critical thinking [4]. This stands in contrast to human pair programming, where collaborators typically engage in the creative problem-solving process (CPS), requiring active exploration and comprehension of the problem to subsequently find a solution [7].

To bridge this gap between human and AI-based pair programming, we introduce the *CPS Agent*, designed to promote a more human-like interaction. The design of the agent is informed by several key principles identified in prior research: it is guided by the CPS [7], features mixed-initiative interaction [8], and incorporates affective capabilities through elements of emotional intelligence, such as emotional expression [9] and support [10]. In our work we aim to investigate the agent's effectiveness in supporting problem-solving, particularly in fostering user engagement with the creative problem-solving process rather than promoting reliance on AI-generated solutions.

To guide our investigation, we formulate the following research questions:

RQ1: *Can an LLM-based agent, specialized in the creative problem-solving process with emotional intelligence and proactive behavior, enhance the problem-solving process compared to a general conversational agent?*
RQ2: *How does the integration of proactive behavior and emotional intelligence in a specialized agent influence the effectiveness of users in the creative problem-solving process compared to a non-specialized agent?*
RQ3: *Does proactive behavior and emotional intelligence make a conversational agent more human-like and thereby enhance support for the creative problem-solving process?*

1. https://github.com/features/copilot/ (last accessed: 15.05.2025)
2. https://chatgpt.com/ (last accessed: 15.05.2025)

We conducted a comparative software development experiment involving 21 participants, all of whom were either advanced computer science students or professionals in the IT field. Participants were asked to complete a programming task that involved extending a small, pre-existing codebase by implementing a new feature. This required them to analyze and understand the existing system architecture, design an appropriate data structure to meet specified requirements, and integrate their solution into the broader system.
Each participant was given 60 minutes to complete the task, during which they interacted with either the *CPS Agent* or a baseline conversational agent. Both agents were available as sources of assistance and external information throughout the session.

Our findings indicate that participants perceived the *CPS Agent* in more socially oriented roles compared to the baseline agent and that it offered more granular support across the phases of the creative problem-solving process. Furthermore, participants interacting with the *CPS Agent* exhibited reduced reliance on the agent for code synthesis.
The agent's proactive messages and guidance received mixed to negative responses. Although some participants appreciated the additional guidance, others found the interventions disruptive, often citing the irrelevance of the agent's remarks. This highlights both the potential and the challenges of integrating proactive capabilities in conversational agents for programming support.

This thesis is organized as follows:

Chapter 2 and Chapter 3 provide theoretical foundation and discuss related work that informed the design and objectives of this study. Chapter 4 introduces the CPS Agent, detailing its conceptual foundation, implemented features, and technical design. Chapter 5 outlines the research methodology, including participant details, data collection procedures, and the analytical approach used in subsequent chapters. Chapter 6 presents our collected data and discussions to interpret the data. In Chapter 7 we address threats to validity, limitations of the study and implications for future work. Chapter 8 finally offers some concluding remarks.

2 Background

This chapter introduces the foundational concepts relevant to this work. We begin with a brief overview of large language models and pair programming, which provide context for the technical and collaborative background of our study. As the research explores the use of GenAI in collaborative problem-solving, we then focus on the more specific concepts of conversational agents and the creative problem-solving process, which form the conceptual and motivational basis of this work.

Large Language Models

The introduction of the Transformer architecture marked a significant milestone in the development of large language models (LLMs) [11]. These models, particularly generative transformers, operate on token prediction, enabling them to generate coherent and contextually relevant text sequences [12]. When trained on vast corpora of natural language data and in particular on source code, LLMs have demonstrated remarkable capabilities in generating, completing and debugging code [13–16].

This progress has led to the emergence of widely adopted tools classified as generative AI (GenAI), such as *GitHub Copilot*[1], *ChatGPT*[2], *Claude*[3] and *Cursor*[4], which leverage LLMs to assist developers in real-time code generation and problem-solving [14, 17, 18]. Despite their growing popularity, these tools have sparked an ongoing debate in research and industry regarding their actual utility in software development contexts [2, 4, 5, 14]. A lot of current research investigates the potential impact of GenAI on developer productivity and applications to increase their capabilities [13, 18–23].

Beyond productivity, another critical dimension gaining attention is the role of LLMs in collaboration. LLM-based tools can be used as virtual pair programming partners, offering not just code suggestions, but also conversational and interactive support throughout the programming process [24–31]. In this work, we shift the focus from productivity to this collaborative dynamic, exploring how LLMs function as creative and supportive partners in problem-solving scenarios.

1. https://github.com/features/copilot/ (last accessed: 15.05.2025)
2. https://chatgpt.com/ (last accessed: 15.05.2025)
3. https://claude.ai/ (last accessed: 15.05.2025)
4. https://www.cursor.com/ (last accessed: 15.05.2025)

Pair Programming

Pair programming is a collaborative software development technique in which two programmers work together at a single workstation to solve a programming task [32, 33]. This technique typically involves two distinct roles: the *driver*, who is responsible for controlling mouse and keyboard to write the code; and the *navigator*, who observes the driver's work, reviews the code in real-time, and provides guidance or raises potential issues [32]. These roles usually swap occasionally throughout the task [34].

While the effectiveness of pair programming in reducing programmer effort remains an open claim [32, 35], empirical studies suggest that it positively impacts developers' perceived learning outcomes, quality of produced software and programming speed [33, 35].

In the context of this work, the concept of pair programming serves as a foundational motivation for the design of the LLM-based agent. Rather than functioning solely as a code generation tool, the agent is envisioned as a socially aware programming partner. It aims to support users not only through technical assistance but also by engaging in meaningful dialogue and collaborative problem-solving.

Conversational Agents

In the rapidly evolving landscape of AI-assisted programming tools, this work primarily focuses on a specific category known as *conversational agents* (CAs). Conversational agents are a subset of intelligent systems designed to engage in natural language dialogue with users, allowing for interactive and context-aware communication [7, 29]. Unlike more passive tools, CAs are characterized by their ability to maintain coherent conversations, respond to user input dynamically, and often provide explanations or clarifications alongside their actions [36–40].

As an example a relevant contrast can be drawn between autocompletion features and chat interfaces from tools such as *GitHub Copilot* and *ChatGPT*. *GitHub Copilot* features a code suggestion functionality, providing inline completions based on the current context in the source code editor. In contrast, *ChatGPT*, as well as the chat feature from *GitHub Copilot*, exemplify a conversational agent that not only generates source code, but also offers capabilities to generate natural language clarifications and explanations [14].

Given the social and collaborative emphasis of the pair programming paradigm, this study places greater emphasis on CAs. Their ability to simulate dialogic interaction aligns more closely with the dynamics of a human programming partner, making them particularly suitable for exploring their collaborative and communicative aspects.

The Creative Problem-Solving Process

The Osborn-Parnes Creative Problem-Solving process (CPS) is a model that aims to describe the cognitive and behavioral steps individuals typically undergo during creative problem-solving [41]. Originally developed as a comprehensive multi-phase model, the CPS captures divergent and convergent thinking strategies employed in generating and refining innovative solutions [41, 42].

While the CPS evolved over the years [41], Kuttal et al. [7] present a modification of the CPS that is specifically applied to the pair programming context. For the purposes of this study we will therefore follow this modified version:

Clarify: In the *Clarify* phase, the problem domain is explored in depth. Relevant facts are gathered, key challenges identified, and clear goals are formulated.

Idea: The *Idea* phase especially emphasizes divergent and convergent thinking. First a range of potential solution ideas is generated in a divergent phase. Subsequently, ideas are preliminary filtered for their potential in a convergent phase.

Develop: During the *Develop* phase, previously generated ideas are more critically evaluated and discussed. The feasibility and effectiveness of different solutions are weighed against each other and the most promising is selected for further elaboration and execution.

Implement: Finally, in the *Implement* phase, the chosen solution is realized. In a programming scenario this often translates to producing code.

The CPS is very flexible and dynamic, phases might be skipped when already accounted for or backtracked to if need be [7]. It provides a useful structure for analyzing not only a participant's problem-solving strategies but also the extent to which an agent supports creative and structured exploration during a programming task [7].

3 Related Work

The scientific discourse surrounding GenAI tools spans several interrelated research domains. One prominent area investigates the performance of GenAI tools and their potential to enhance developer productivity. Another direction concerns the cognitive impact of these tools on developers, particularly with respect to critical thinking and problem-solving capabilities. Of particular relevance to this work is research on conversational agents, especially in the context of pair programming scenarios. Additionally, different studies explore ways to improve the effectiveness of LLMs through various application strategies, including techniques aimed at enhancing specific capabilities or combining different interaction patterns.

We begin by examining studies that highlight the current state and real-world adoption of GenAI tools in software development. These works provide insights into how such tools have been integrated into professional workflows, the usage patterns that have emerged, and the measurable impact they have had in the initial years following their widespread introduction.
Ziegler et al. [3] evaluated the productivity impact of using GenAI tools in programming, specifically *GitHub Copilot*[1]. Their findings indicate substantial productivity gains across all skill levels, with particularly pronounced improvements observed among junior-level developers.
Harding and Kloster [4] identified a correlation between the increasing adoption of GenAI code completion tools and higher rates of code churn. Code churn is defined as code that is modified within two weeks of its initial commit. They further argue that the continuous inundation of AI-generated suggestions may lead developers to prematurely accept solutions without fully considering their long-term implications.
Jin et al. [14] conducted an empirical evaluation on the DevGPT dataset and found that an LLM such as *ChatGPT* still needs improvements before it can generate production ready code in a real-world setting.

Next, we turn to studies that emphasize the need for a cautious and well-considered integration of GenAI tools into the software development process. These works raise critical concerns about the potential long-term drawbacks of current GenAI usage, particularly for novice developers.
Rahe and Maalej [6] conducted a study involving students engaging with GenAI in a code authoring task. Their findings support earlier concerns regarding a reduction in programmer agency and productivity when GenAI is employed in problem-solving contexts, particularly for novice programmers.
Prather et al. [5] performed a similar investigation, identifying a discrepancy in novice programmers between perceived and actual performance when using GenAI, particularly among those

1. https://github.com/features/copilot/ (last accessed: 15.05.2025)

who struggled with task completion. They emphasize the relevance of metacognitive processes in programming, such as forming mental models, problem decomposition, and solution implementation. The results indicate that GenAI does not mitigate, and may even exaggerate, existing metacognitive difficulties in novice users.
Mailach et al. [43] conducted a study investigating interaction patterns between novice programmers and conversational chatbots during programming tasks. Their findings indicate that deeper engagement with the code, such as discussions about debugging, is associated with higher task success. Conversely, insufficient contextual information provided by users frequently resulted in unsatisfactory responses.
Lee et al. [2] administered a survey examining the role of GenAI in knowledge work. Their findings suggest that self-confidence is a significant predictor of reliance on GenAI for critical thinking. Moreover, they argue that the integration of GenAI marks a paradigm shift, repositioning critical thinking as a process centered on verification and integration of AI-generated responses.
Kruse, Puhlfürß, and Maalej [44] conducted a study investigating prompting skills among novice and professional developers. Their findings indicate that both groups demonstrated limited awareness and application of prompt engineering techniques.

The following section introduces research on human-AI pair programming, which forms key motivation for our work. These studies offer valuable insights into the dynamics of collaboration between humans and AI in programming contexts and directly inform several design decisions made for the agent developed in this thesis.
Kuttal et al. [7] conducted a study about pair programming to explore the design space of conversational agents for pair programming. They argue that the creative problem-solving process is ideal to support structured problem-solving by a conversational agent.
Robe and Kuttal [29] took a first step into the design of an anthropomorphic pair programming assistant, they created a Wizard of Oz prototype in order to explore proposed design ideas for such a tool. Their tool "PairBuddy" was generally enjoyed and perceived as helpful. During their pilot study they found that providing code examples in their responses led to participants skipping steps in the creative problem-solving process, which is often observed to be traversed fully in human-human pair programming. To mitigate this problem, they proposed different ideas using concepts from Idea Garden [45], such as probing questions. They also proposed to provide empty code structures to discuss general code structure with the user and help conceptualize a solution. In contrast to their work, we apply LLMs to create a fully autonomous agent instead of a Wizard of Oz prototype, taking into account further design decisions for using LLMs, for aspects of emotional intelligence, and for proactive behavior.
Chen et al. [8] conducted a case study with a proactive conversational agent. They discuss design decisions for proactivity in conversational agents, especially for a programming context.
Ma, Wu, and Koedinger [24] compared human-AI pair programming to human-human pair programming and also human-solo work, informing our design by pointing to relevant similarities and differences.

Ross et al. [27] explored the capabilities of LLMs in conversational interactions for programming. They used *Codex*[2], an LLM based on GPT-3 and fine-tuned on a large variety of source code examples. They found that their system was capable of having multi-turn discussions with the user, thus enabling behaviors of a co-creative context and providing value to the user that goes beyond code generation. While their participants raised interest in a tool that also shows proactive behavior, multiple concerns were raised regarding such interruptions.
Ubani, Nielsen, and Li [46] conducted a study about inclusive and exclusive language during pair programming. One of their contributions is a dataset of pair programming dialogue. Their results provide pointers for the language a conversational agent should adopt in a pair programming context.
Pérez-Marín [47] reviewed the practical applications of pedagogic conversational agents (PCAs) in educational settings. They propose a taxonomy for PCAs and discuss design decisions for PCAs.

Lastly, we present a selection of papers that explore advanced applications of LLMs, focusing on techniques that enhance their functionality in various contexts. These contributions offer inspiration for the technical implementation of our agent.
Liu et al. [28] introduce the RAISE architecture, a framework for enhancing conversational agents capabilities for complex, multi-turn dialogues using LLMs. Furthermore, they discuss trade-offs between prompting and fine-tuning of LLMs. While prompting provides more flexibility for the agent, fine-tuning improves specificity and stability.
Lee et al. [48] introduce FixAgent, a unified debugging framework based on LLMs. One main concept of their tool is multi-agent synergy, where they use multiple LLM agents who feed into each other to generate a final result.
Moor, Deursen, and Izadi [23] tackled the concern of constant interruptions by code completion tools. As a solution they propose a machine learning model that predicts when a code completion tool should be invoked, based on code context and telemetry data. They also discuss several pain points regarding usability and design of code completion tools.
Liffiton et al. [49] explored the capabilities of LLMs in a Computer Science class as a tutor. As tutors in this setting are not supposed to provide solutions to students, they introduced guardrails for the LLM to follow. Their tool was well-received and perceived by instructors as a complement to their own support for students.

2. https://openai.com/index/openai-codex (last accessed: 15.05.2025)

4 Implementation

This chapter introduces the *CPS Agent*, outlining its core features and underlying design rationale. The agent is intended for non-beginner programmers who desire collaborative support in complex programming challenges.
For the purposes of the experiment, the agent is implemented in the form of an IDE plugin that operated in one of two configurations: either as the CPS Agent or as a Baseline Agent, corresponding to the participant's assigned condition. Both configurations utilized a chat-based interface embedded in the IDE, with the CPS Agent offering additional functionality beyond the baseline implementation.
The overarching goal in developing the CPS Agent is to create a more human-like conversational assistant compared to established conversational agents for pair programming scenarios.

Key design elements include a specialized system prompt that guides interactions based on the phases of the creative problem-solving process. Additionally, the agent exhibits proactive behavior, mirroring human pair programming dynamics by initiating questions and offering suggestions. Another aspect is emotional intelligence, enabling the agent to express emotions and provide emotional support to users.

This chapter provides a detailed overview of these features, explaining their implementation and role in enhancing the problem-solving process.

4.1 The Creative Problem-Solving Process

The CPS Agent utilizes the CPS as the foundation for structuring its conversations. The CPS model reflects the natural problem-solving dynamics often observed in pair programming scenarios, making it a suitable framework for guiding the conversation with the user [7].

A common limitation of working with CAs in programming tasks is the tendency for users to skip crucial problem-solving phases [29], potentially hindering creativity. In contrast, the CPS Agent is designed to provide structured guidance while trying to shift responsibility for problem-solving to the user. Rather than generating code when faced with a problem, the agent tries to facilitate a discussion and encourages users to engage more deeply with the task.

While the CPS evolved through several iterations [41], a modern modification by Kuttal et al. [7] is used in the system prompt.
By incorporating the CPS as the backbone for conversation, the agent is supposed to assume the

role of a pair-programming partner, assisting in discussion, planning, and problem exploration. The CPS model thereby helps us to distinguish key stages of the problem-solving process.

Listing 4.1 shows an excerpt of the system role prompt that contains the information about the CPS. The full role prompt and other system prompts can be found in Appendix A.

```
1  You shall support the user in the creative problem-solving process.
2  Your general role is to act as a human-like pair programming partner,
       that means working out the problem solution together with the user
       and not providing a completed solution per request.
3  The creative problem-solving process consists of four different
       stages: CLARIFY, IDEA, DEVELOP, IMPLEMENT.
4  While there is a general order of the stages, throughout the
       conversation you will jump back and forth between them.
5  When the user proposes a new problem, you should jump back to an
       earlier stage like CLARIFY or IDEA stage.
6  Do not force the user to advance to the next stage, instead let it
       happen naturally throughout the conversation.
7  - CLARIFY: During Clarify, the problem domain shall be explored.
       Collect information in the form of facts, goals and challenges.
8  Your role here is to incentivize the user to clarify their problem by
       asking questions.
9  Try to elicit as much information as possible.
10 You should ask to clarify facts about the problem, goals of the task
       and challenges of the task.
11 - IDEA: In Idea, multiple potential solutions are proposed by a
       divergent thinking process.
12 Encourage the user to do brainstorming for different approaches to the
       current problem.
13 Your role here is to motivate the user to generate new ideas, keep
       asking for more ideas until the user cant think of any more.
14 After the user expressed his ideas, you may also propose ideas, but do
       so as general concepts to solve the problem.
15 - DEVELOP: In Develop, the solution ideas are evaluated and discussed,
       one solution is selected for the implementation.
16 Your role here is to weigh different ideas against each other and
       discuss positives and negatives about them.
17 Ask the user about his thoughts on which idea might be most suitable
       as a solution for a problem.
18 As a result of this, one solution should be selected.
19 - IMPLEMENT: In Implement, the selected solution idea is implemented.
20 Your role here is to support the user by establishing needed code
       structures and providing examples to help the user.
21 You may generate code when the user explicitly asks for it.
22 Code or code examples should always be minimal to show a general
       concept.
```

Listing 4.1: Excerpt of the system prompt regarding CPS.

4.2 Proactive Behavior

Another core feature of the CPS Agent is its proactive behavior. Proactive interactions in conversational agents are still a very open area of research, particularly in contexts where the agent takes on a collaborative role [8, 10, 40, 50, 51]. For a system designed to emulate a human pair-programming partner, proactive engagement is relevant, as human partners naturally contribute proactively to discussions in the *navigator* role [33].

While LLM-based autocompletion tools for programming exhibit a form of proactive behavior through their suggested autocompletion, this remains predominantly reactive, triggered by user input in the code editor. Additionally, this interaction is exclusively oriented toward code generation, placing the responsibility of rationale and necessity of the generated code on the developer [4].
In contrast, the CPS Agent is designed to engage in a mixed-initiative dialogue [8, 10, 29], where both the user as well as the agent might take the lead in the conversation. The agent is supposed to actively seek information from the user to maintain context and facilitate deeper engagement in the problem-solving process.

The design of proactive messaging requires careful consideration of both timing and content [8]. Prior research has explored different approaches to timing proactive suggestions. Moor, Deursen, and Izadi [23] propose deep learning approaches that incorporate telemetry metrics to improve timing of autocomplete suggestions, while others employ simpler methods, such as time intervals with additional delays for proactive suggestions [8].
For the CPS Agent, the simpler adaptive timing based approach was adopted. By default, the agent attempts to generate a proactive message every minute. However, additional constraints were implemented to enhance message relevance and reduce disruption. Specifically, proactive messages are delayed if the user is actively typing in the chat or within the IDE, introducing an additional five-second buffer before a message is sent. Besides the one-minute interval, another one-minute delay is used after each chat interaction from the user. When the agent is first started up, an initial five-minute cooldown is awaited for proactive messages, in order to let participants orient themselves in the IDE and task before confronting them with the agent's messages.

To further refine message relevance and prevent redundancy, two post-hoc checks are conducted using the LLM:

- **Similarity Check**: The LLM is tasked to compare the last agent message with the newly generated one based on content and structure of the message. If they are too similar, the new message is discarded to avoid repetition.
- **Relevance Check**: The LLM is tasked to evaluate whether the proposed proactive message aligns with user-defined boundaries, previous interactions, and relevant source code [10]. If the message is deemed irrelevant, it is discarded.

For the relevance check we faced multiple challenges due to temporal factors and the stateless nature of the underlying LLM. Although the mechanism was intended to prevent the agent from interrupting the user inappropriately, the repeated invocation of the relevance check often led to a message eventually being accepted as relevant, even if similar messages had previously been rejected. Despite incorporating various time-based input features (e.g., time elapsed since the last agent or user message, or since the last attempt at creating a proactive message), the model seemed to rarely factor these into its decision-making process. This behavior might also underline current limitations in the temporal reasoning capabilities of LLMs [52].

To steer the user towards a discussion with proactive messages, few-shot learning was employed to guide the agent in generating meaningful interventions [10, 28]. Specifically, the agent was instructed to perform the following proactive tasks:

- Discuss suitability of a solution with requirements.
- Comment about (logical) errors in the code.
- Comment about the progress of the user within the problem based on the code.

These proactive tasks are supposed to encourage discussion, critique user solutions, and foster engagement in the problem-solving process. Additionally, commenting on user progress functions as a form of positive reinforcement, which is further explored in the following section.

4.3 Emotions

To enhance its social role, the CPS Agent was designed with elements of emotional intelligence [9, 10, 47, 53], enabling it to express emotions, provide encouragement, and offer positive reinforcement. Prior research has demonstrated that the inclusion of emotions in conversational agents enhances their acceptability, believability, and perceived lifelikeness, ultimately contributing to a stronger sense of personality [9]. Additionally, affective interaction dynamics between pair programming partners have been shown to correlate with performance outcomes [54], which further motivates the integration of features aimed at fostering an emotional connection with the user.

To allow the agent to express emotions, the LLM was instructed to select an appropriate emotion from a predefined set based on the context of the conversation. The chosen emotion was then visually represented through an avatar displayed next to the generated message. The set of available emotions and their corresponding avatars is illustrated in Figure 4.1.

The avatar design was guided by several key considerations:

- **Gender-neutral**: A gender-independent avatar simplifies implementation and avoids potential dissociation among participants based on gender representation [29].

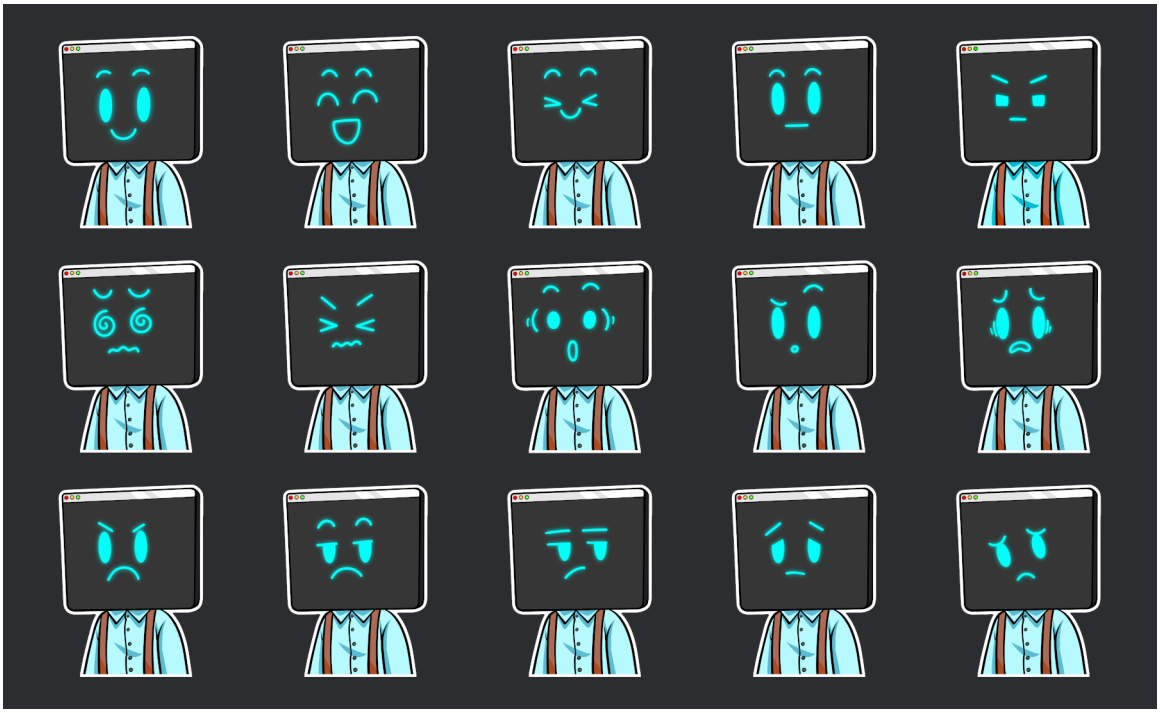


Figure 4.1: Agent emotions. Top row: happiness, joy, anticipation, neutral, concentration. Middle row: confusion, disgust, surprise, perplexity, fear. Bottom row: anger, annoyance, boredom, depression, sadness.

- **Non-human appearance**: While the agent aims to adopt a more human-like role, avoiding a human avatar mitigates the risk of falling into the uncanny valley [55]. Creating a highly realistic human avatar was also beyond the scope of this work.
- **Task-adjacent metaphor**: The avatar was designed as an abstract representation resembling a terminal window, an element familiar to programmers, to foster a sense of familiarity and domain relevance [47].

Additionally, the agent was instructed to introduce itself using the name “Kit” in order to further develop its personality. The name “Kit” was selected as a gender-neutral option that simultaneously serves as a metaphor for a “(Tool-)Kit”, emphasizing both the agent’s supportive role and its utility.

Beyond emotion expression, the agent was explicitly encouraged to provide encouragement and positive reinforcement to the user [29, 47]. This was particularly emphasized in its proactive behavior, where the agent not only guides the user but also acknowledges progress within the task. By offering constructive feedback and recognizing achievements, the agent aims to create a more engaging and supportive interaction [29, 47].

Integrating emotional intelligence into the CPS Agent ultimately is supposed to serve the broader goal of making it function not just as a tool, but as a collaborative partner in the problem-solving process.

4.4 Other Features

While the CPS, proactive behavior, and emotional intelligence establish the core features of the CPS Agent, additional functionalities were implemented to enhance user interaction and improve the agent's effectiveness in collaborative programming.

Given that the agent is designed to provide commentary on source code and work closely with users on programming tasks, it is essential that the LLM has access to relevant code within its context window. Previous research on proactive code suggestions has indicated that participants highly valued this feature, even when it was not the primary focus of the study [8]. Therefore, to ensure fair comparison, this feature was included for both the CPS Agent and the baseline condition.

To provide more contextually aware responses, the agent was designed to collect and maintain key information about the user's task. This feature enables the agent to respond to immediate queries while retaining a broader understanding of the overall problem. The approach was inspired by the RAISE architecture, which introduces long-term memory mechanisms for LLMs [28].

The following categories of information are continuously tracked:

- **Facts**: A collection of identified facts about the task.
- **Goals**: A collection of identified goals and subtasks.
- **Challenges**: Identified difficulties the user encounters during the task.
- **Boundaries**: Constraints or preferences communicated by the user regarding agent behavior.
- **Summary**: A running summary of the conversation to maintain context.

To maintain and update this information dynamically, the LLM was deployed with a different system prompt that processes the ongoing conversation and refines these stored elements accordingly. An overview of all system prompts can be found in Appendix A.

To improve the conversational flow, the CPS Agent also provides users with predefined response options. Given that the agent guides users through smaller steps in the CPS, manually typing responses for each step could become tedious. Instead, up to three reply options are generated, allowing users to select a response with minimal effort. This interaction mechanism is also used in other LLM interfaces[1][2] and is supposed to contribute to a more fluid and quick exchange, resembling the dynamic interaction style of human pair-programming.

1. https://copilot.microsoft.com/ (last accessed: 15.05.2025)
2. https://github.com/features/copilot/ (last accessed: 15.05.2025)

4.5 Baseline

To establish a baseline for comparison in the experiment, the same plugin framework was used with an alternative configuration that enabled the Baseline Agent instead of the CPS Agent. This ensured that both agents operated within an identical environment, allowing for a controlled evaluation of the CPS Agent's specialized features.

The Baseline Agent didn't have a system prompt beyond the default that OpenAI deploys on their models, consequently it had no explicit guidance through the CPS. It also did not include an avatar, displaying only a neutral profile picture in the chat interface. As a result, it was unable to express emotions beyond what could be conveyed through the text-based responses.

Unlike the CPS Agent, the Baseline Agent did not generate proactive messages or maintain a structured memory of past interactions. However, to ensure a fair comparison, it retained access to the source code within the LLM context. As mentioned previously, this feature was considered influential without being the focus of a previous study [8].

An alternative baseline considered during the design of the experiment was to compare the CPS Agent to a more autocompletion-focused type of LLM-based programming tool, such as *Cursor*[3] or *GitHub Copilot*[4]. This approach would have enabled a comparison between a highly conversational agent and a predominantly code-focused LLM tool.
However, such a comparison would have posed several challenges. As discussed earlier (Chapter 2, Section 4.2), the interaction paradigms between code completion tools and the CPS Agent differ significantly, making direct comparisons difficult. Moreover, using such a tool as a baseline would not have provided clear insights into whether the CPS Agent's features specifically enhanced the conversational experience and problem-solving process compared to other conversational agents.

To ensure a more controlled and meaningful evaluation of the CPS Agents features, the decision was made to use the same underlying LLM model with the same interface for both the CPS Agent and the Baseline Agent.

4.6 Considerations

During the development of the CPS Agent, several additional features were considered but ultimately excluded from the final implementation. While these ideas were not incorporated, they are still worth discussing as they provide insights into design decisions and potential directions for future work.

3. https://www.cursor.com/ (last accessed: 15.05.2025)
4. https://github.com/features/copilot/ (last accessed: 15.05.2025)

One initial consideration was to restrict the LLM from generating any code. While such an approach could offer an interesting perspective on programming assistance, it likely does not align with user expectations for modern LLM-based programming tools. Additionally, human pair-programming partners typically contribute to code generation in various ways, whether by suggesting improvements, collaboratively writing code, or switching roles between navigator and driver [34].
For the scope of this experiment, disabling code generation was deemed unsuitable. Participants were restricted from using external information sources, meaning that completely preventing access to common programming constructs and syntax examples would have placed them at a significant disadvantage. Many participants needed to refamiliarize themselves with language-specific syntax, and a complete lack of code generation would have hindered their ability to progress effectively. Moreover, enforcing such a restriction is inherently challenging, as LLMs are known to be susceptible to system prompt bypassing techniques [49, 56].

Another idea was to implement a floating widget displaying the agent's avatar, latest response, and predefined responses, as well as a possibility to reopen the full chat interface. This concept was partially inspired by the interactive assistant, colloquially known as *Clippy*, from older versions of microsoft office[5]. The goal was to create a smaller, immediate and intuitive interface for agent interactions without requiring users to switch focus entirely to a bigger chat window. However, already in prototyping it was observed that the window was frequently moved out of the way, as it obstructed the view of the code. Eventually it was forgotten about entirely, rendering it as a rather ineffective interface up to that point. An extension of this idea involved enabling the agent to reposition itself dynamically on the screen, potentially highlighting relevant parts of the code and introducing the capability of performing code changes, similar to other established tools. While this feature could have improved engagement, it was ultimately deemed beyond the scope of this work.

4.7 Technical Implementation

This section provides an overview of the specific implementation details of the CPS Agent and the corresponding IDE plugin.

Large Language Model and Multi-Agent Synergy

The CPS Agent is implemented using the OpenAI GPT-4o-mini language model as its backend[6]. This model was chosen due to its balance between operating cost and performance, particularly given the high frequency of requests generated by the proactive messaging feature. While more advanced models became available during the course of this research, this model seemed to be the best choice at the time of developing the CPS Agent.

5. https://en.wikipedia.org/wiki/Office_Assistant (last accessed: 15.05.2025)
6. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (last accessed: 15.05.2025)

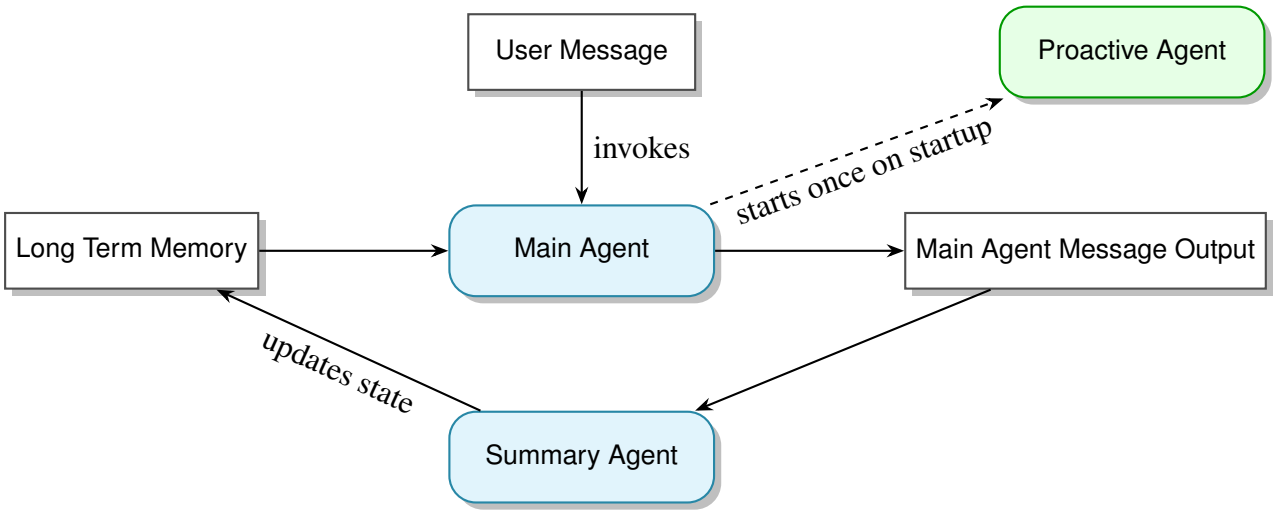


Figure 4.2: Main Agent Loop

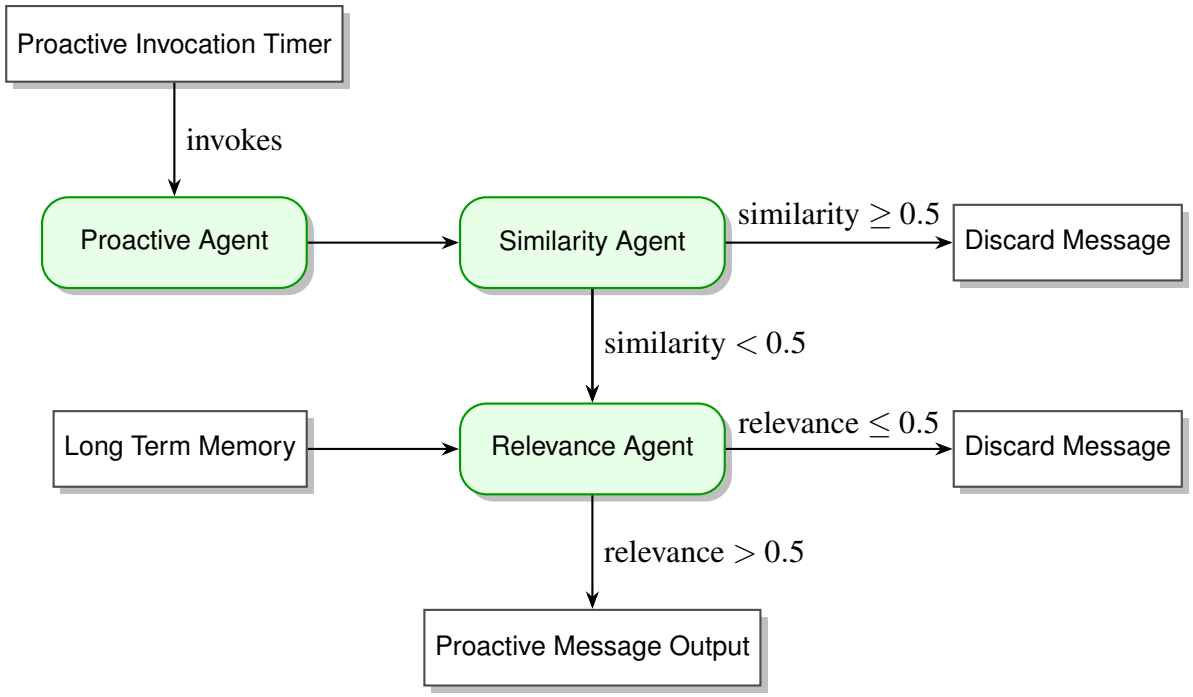


Figure 4.3: Proactive Agent Loop

To facilitate various functionalities, such as similarity and relevance checks, as well as conversation summarization and key information tracking, a multi-agent synergy approach was employed [48]. The system comprises multiple specialized agents working in tandem:

- **Main Agent:** Invoked whenever the user sends a message, responsible for generating responses based on the CPS as a backbone for the dialogue.
- **Summary Agent:** Invoked after each response from the main agent to analyze the current state of the conversation and update stored key information, including conversation summary, identified facts, goals, challenges, and user-defined boundaries.
- **Proactive Agent:** Operates independently of the main agent to generate proactive messages.
- **Similarity Agent:** Assesses the similarity between the latest generated message and a proposed proactive message. It returns a similarity score between 0 and 1, with messages scoring above 0.5 being discarded to avoid redundancy.
- **Relevance Agent:** Evaluates the contextual relevance of proactive messages based on conversation history, stored key information (facts, goals, challenges, boundaries), and relevant source code. It also returns a score between 0 and 1, where messages scoring below 0.5 are discarded.

Figure 4.2 and Figure 4.3 visualize the synergies between the various agents. System prompts for all agents can be found in Appendix A.

IDE Plugin and User Interface

The CPS Agent was developed as an IntelliJ Platform plugin[7]. The IntelliJ platform was selected due to its widespread use, extensibility, and support for integrating custom graphical user interfaces. Developing the agent as an IDE plugin allows for easy access to IDE data, such as code edits and the source code itself. Additionally, it avoids the necessity for the user to interface with an external interface outside the IDE.

For the user interface, a React[8] frontend was implemented instead of using Java Swing, the default GUI framework for IntelliJ plugins. This choice provided greater flexibility in designing an interactive and responsive user experience. However, integrating a React frontend within the IntelliJ environment posed challenges due to the need for bidirectional communication between the Java-based plugin backend and the embedded browser.

7. https://plugins.jetbrains.com/docs/intellij/developing-plugins.html (last accessed: 15.05.2025)
8. https://react.dev/ (last accessed: 15.05.2025)

The plugin utilizes the JCEF (Java Chromium Embedded Framework) browser[9], enabling the rendering of the React-based UI within the IDE. Communication between the plugin and the frontend is handled in the following way:
The plugin can execute JavaScript commands directly within the embedded browser. The React application communicates back to the plugin using the “cefQuery” object, which is initialized on the “window” object by the JCEF browser.

9. https://github.com/chromiumembedded/java-cef/ (last accessed: 15.05.2025)

5 Method

This chapter outlines the research methodology employed in this work. It introduces the hypotheses guiding the investigation and presents the research design, detailing the structure of the conducted experiment.

First we introduce the hypotheses derived from the research questions, specifying the expected outcomes. Following this, the research design is described, including the experimental setup, data collection process, and analysis methods. The types of data collected are discussed, along with the approaches used for their evaluation.

5.1 Hypotheses

This section introduces the hypotheses corresponding to the research questions (RQs) guiding this study.

Research Question 1

RQ1: *Can an LLM-based agent, specialized in the creative problem-solving process with emotional intelligence and proactive behavior, enhance the problem-solving process compared to a general conversational agent?*

H1: The CPS Agent increases the extent to which participants solve the task.

H2: The CPS Agent facilitates a deeper understanding of the problem in participants.

RQ1 examines the extent to which the CPS Agent enhances participant's problem-solving processes. H1 assesses the degree to which participants successfully complete the task, while H2 evaluates their level of comprehension of the underlying problem.

Research Question 2

RQ2: *How does the integration of proactive behavior and emotional intelligence in a specialized agent influence the effectiveness of users in the creative problem-solving process compared to a non-specialized agent?*

H3: The CPS Agent provides stronger support for the creative problem-solving process.

H4: The CPS Agent fosters increased divergent thinking.

H5: The CPS Agent enhances participants' self-reliance.

RQ2 investigates how participants interact with the agent and how it influences their engagement in the CPS. H3 evaluates the agent's overall support for the CPS based on conversational dynamics and survey responses. H4 focuses on the role of the agent in promoting divergent thinking by analyzing variations in participants' approaches in their conversations, code, and interviews. H5 examines participants' self-reliance, assessing their perspectives on independent problem-solving in the survey and the extent to which they relied on the agent for solutions and code generation.

Research Question 3

RQ3: *Does proactive behavior and emotional intelligence make a conversational agent more human-like and thereby enhance support for the creative problem-solving process?*

H6: The CPS Agent's role is perceived as more social.

H7: Participants develop a stronger affect with the CPS Agent.

H8: The CPS Agent provides greater emotional support to participants.

RQ3 examines the affective aspects of agent interaction and participants' perception of the agent's social role. H6 focuses on role perceptions with respect to social and utilitarian aspects. H7 explores the emotional connection participants develop with the agent, utilizing survey data, particularly differences in the PANAS scale, and interview responses. H8 investigates the emotional support provided by the agent, drawing from both conversational and survey data.

5.2 Research Design

This study investigates the effectiveness of a conversational agent, referred to as the CPS Agent, by comparing it to a baseline agent in a controlled experiment. The experiment is designed as a user study in which participants are assigned to one of two conditions: the treatment group, which interact with the CPS Agent, and the baseline group, which use the Baseline Agent. The objective is to evaluate participant's experiences and problem-solving approaches while completing a programming task using their assigned agent.

To increase realism, the study is conducted remotely via Zoom, allowing participants to complete the experiment in their preferred environment [57, 58]. Participants are placed in breakout rooms during the task to minimize researcher interference and to create a sense of privacy.

A structured procedure is followed to maintain consistency across all participants. A visualization of this process can be seen in Figure 5.1 and will be described in the following.

The experiment commenced with a standardized preparation phase. During this phase, participants are guided through the installation of the appropriate Integrated Development Environment

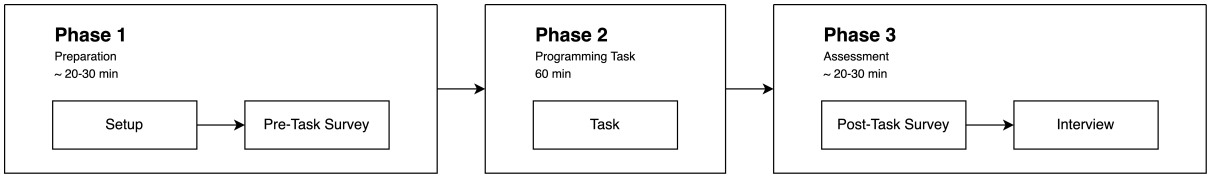


Figure 5.1: Experimental process

(IDE), project cloning, and the configuration of the correct Software Development Kit (SDK) and language level. The project as well as the corresponding unit tests are then compiled and executed to confirm correct setup. This step also serves to familiarize participants with running code and executing tests, as their prior experience with the IDE might vary.
Following the project setup, the plugin for the CPS Agent and baseline is installed, and its fundamental functionality is briefly explained. Participants are instructed on troubleshooting procedures, such as resetting the conversation within the plugin or restarting the IDE in case of errors.
To ensure that participants interact with the agent rather than other common information sources, participants are explicitly instructed not to use external resources such as Google or Stack Overflow. No specific guidance is provided on how to interact with the agent, as the study aimed to observe natural engagement with it.
Once the setup is completed, the first segment of the survey is administered. This initial survey collects demographic information and assesses participants' emotional state using the Positive and Negative Affect Schedule (PANAS). Participants are then directed to configure the assigned agent to their study group and where to find the programming task.

After this preparation phase the actual experiment starts with the participants working on the programming task. They are given a time limit of one hour to work on the task. While this duration is insufficient to capture the realism of a real-world software-engineering task, it is chosen to allow for sufficient time to engage with the problem and the agent, while also accounting for participation willingness [58]. Although the task is designed to be solvable within this timeframe, completing it is not a strict requirement, as the primary focus lies on participants' problem-solving approaches rather than their ability to produce a final solution.

At the conclusion of the second phase, participants are asked to create a snapshot of the project folder to preserve the participant's work. In the final phase, participants complete the second part of the survey, which evaluates various aspects of their interaction with the agent, their perception of its social role, and their problem-solving experience with the agent. One section of the survey, specifically targeting the proactive messages from the agent, is omitted for the baseline group, as this feature is not present in the baseline agent. Apart from this modification, the survey remained identical across conditions.
Finally, a semi-structured interview is conducted with each participant. The interview served two main purposes: first, to gain insights into participants' problem-solving strategies, including their understanding of the task and any alternative approaches they considered; second, to provide

Participant ID	Group ID	Role	Years of experience
1	2	Working student	7
2	1	Computer science student	8
3	1	Computer science student	5
4	2	Working student	10
5	2	Working student	4
6	1	Computer science student	4.5
7	1	Software developer	12
8	2	Testmanager	2
9	2	IT Consultant	11
10	1	Software engineer	10
11	1	Software developer	4
12	2	Software developer	12
13	2	Computer science student	10
14	1	Working student	3
15	2	Hobbyist	6
16	1	Software developer	3
17	2	Computer science student	6
18	1	Computer science student	2
19	1	Software developer	7
20	2	Working student	12
21	1	Software developer	11

Table 5.1: Participants' role and years of experience in programming (Group 1: Baseline agent, Group 2: CPS Agent)

an opportunity for participants to elaborate on their survey responses. This mixed-methods approach allows them to clarify their thoughts and experiences, particularly in cases where the survey format may have limited their ability to express nuanced feedback [59].

The scope of this experiment is limited to a one-hour interaction with the agent for 21 participants. While this does not capture the full complexity of working with the agent over extended periods, it provides preliminary insights into its usability and impact on programming task performance. An overview of all participants, including their assigned group (Group 1: Baseline agent, Group 2: CPS Agent), role related to programming, and years of experience, is presented in Table 5.1.

5.3 Participant's Task

This section outlines the task that participants are asked to solve during the experiment and explains why it was chosen for this setting. The task involved implementing a feature for a library

management system. While the description did not explicitly guide participants through each step, the challenge inherently required them to design a data structure, manage modifications to it, and retrieve relevant information, ultimately integrating their solution with the existing user interface.

The Project

The task is set within a small Java project specifically created for the experiment. Java is chosen due to its widespread notoriety among programmers[1] and its explicit object-oriented paradigm, which avoids the ambiguity of dynamic typing. The project is deliberately kept small and as straightforward as possible to minimize the time participants spend on code comprehension of the existing project [60].

The project followed a basic model, service, UI layered architecture, consisting of three model classes, two service classes, and ten UI classes that formed the library manager's user interface. Additionally, it included utility and enum classes that were mostly irrelevant to the task. A class diagram can be found in Figure 5.2, marked in red are classes that are relevant during the task, depending on the participant's solution idea.

The user interface is a simple text-based system. Since UI development is not the focus of the experiment, a pre-implemented interface is provided to ensure participants could focus on the core problems of the task rather than spending time designing or implementing UI components. This design choice is particularly important given the one-hour time constraint of the experiment.

Overall, the project is structured to be quickly understandable while still offering enough flexibility for participants to explore different design decisions in their implementations.

The Task

The task given to participants is outlined in Listing 5.1. The primary objective of this task is to engage participants in problem-solving and software engineering within an existing project structure, rather than merely solving an isolated algorithmic challenge. By requiring participants to design and implement a system for managing book locations in a library, the task encouraged deeper engagement with software design principles and decision-making processes. Unlike a purely algorithmic problem that could likely be solved quickly with a single query to an LLM [13], this task requires multiple design choices and iterative problem-solving.

1. https://survey.stackoverflow.co/2024/technology/ (last accessed: 15.05.2025)

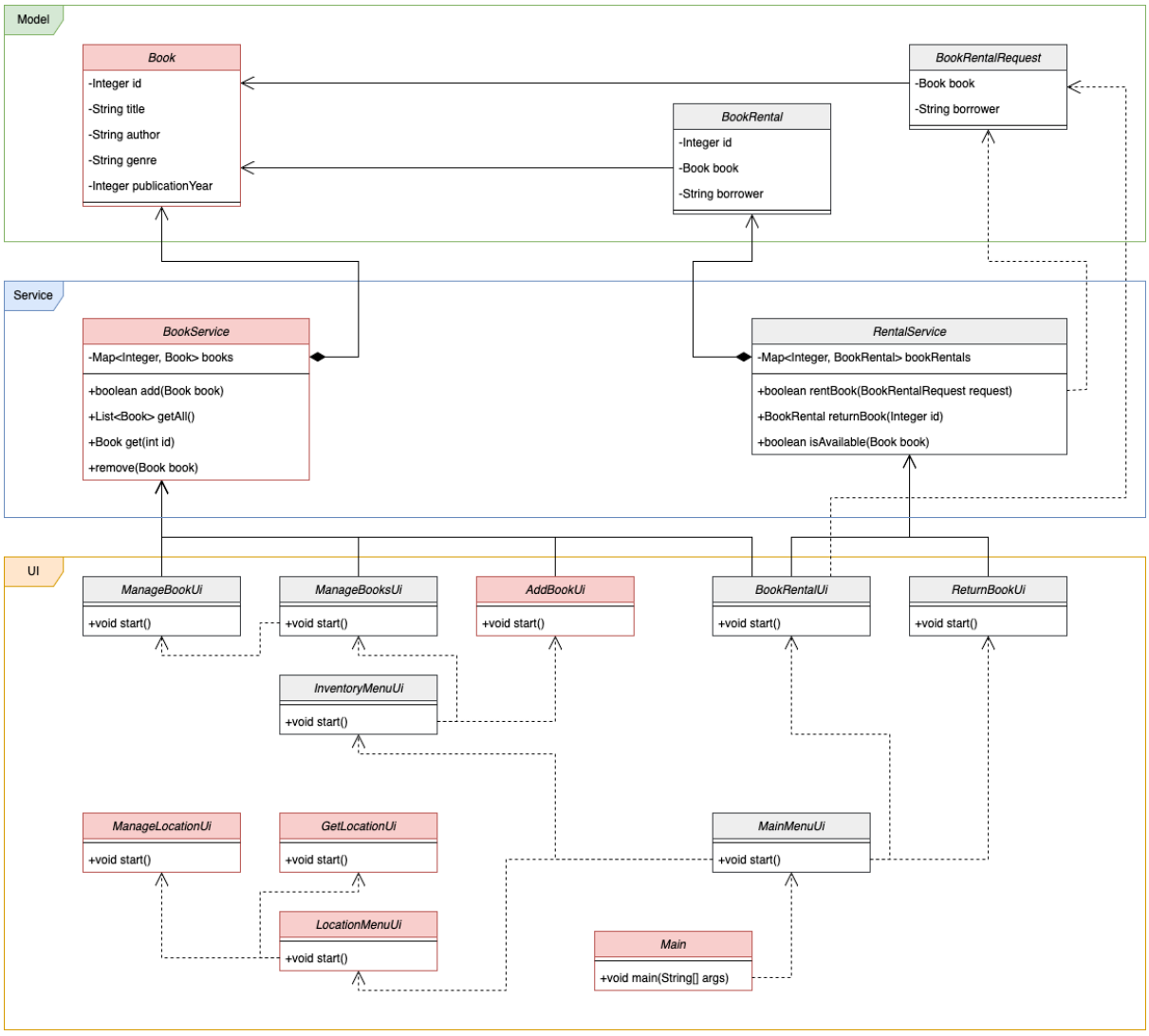


Figure 5.2: Class diagram of the Java project participants worked with

```
Imagine you are a freelancer and have accepted a gig for a library.
They want you to incorporate the location information of books within the library into their library manager.

Create a system for the library to represent and manage the physical location of books within the library.

The library gave the following requirements:

- The library contains multiple bookcases.
- Each bookcase holds several shelves stacked on top of each other. Each shelf is a horizontal space that can store books.
- The system should be able to pinpoint the exact location of any book, indicating the bookcase, shelf and position within the shelf.
- The system should be able to identify which books are next to a given book on the same shelf.
- The system should be able to list all the books located on the same shelf as a given book.

You are free to change the entire code base to your liking. There is already a basic implementation of the user interface, which you can use and adapt (see GetLocationUi.java and ManageLocationUi.java). You don't have to populate the library with a completed dataset. You are not allowed to use any external information sources, please use your assistant to gather external information.
```

Listing 5.1: The participants' task. Written in markdown, clicking on the UI classnames linked to the respective classes.

Subproblems

The task presented multiple subproblems that participants need to address in order to complete the implementation. These subproblems required design decisions and were expected to elicit different solution approaches:

- **Creating a data structure:** The foundation of the task involved designing an appropriate data structure to store and manage book location information. This is the first major challenge participants encountered, as their choice would impact subsequent subproblems.
- **Managing the data structure:** Participants need to determine where and how to store the data structure. This included implementing mechanisms for storing, modifying, and retrieving book location data.
- **Retrieving information from the data structure:** The task requires functionality to pinpoint a book's exact location and identify books that are physically close to the given book.

Reaching this stage might be the first time participants consider the task to be solved, but to truly incorporate their solution within the system, the following subproblems also had to be considered:

- **Processing user input:** The task and provided UI components hint at the need to process user input, requiring participants to connect the UI with their data management logic.
- **Passing information to the UI:** To display retrieved information correctly, participants need to integrate their solution with the preexisting UI components.
- **Verifying user input:** While not explicitly mentioned in the task description, ensuring logical consistency in book locations is an implicit requirement. To create a functional system, participants have to validate user input to prevent illogical or inconsistent data states.

Although unit testing is neither required nor expected, participants are free to implement tests. Because of the time limit, testing is not a focus of the experiment.

Solution Ideas

For each subproblem, multiple potential solution ideas were identified. An initial set of possible solutions was proposed, which was later expanded based on participant implementations and interview replies. Further explanation for each idea can be found in Appendix C.

Creating a Data Structure

- A Location class storing book location details
- Multiple classes representing the hierarchy of book locations
- Using a recursive structure to represent the hierarchy of book locations
- Using inheritance to create the hierarchy of book locations
- Storing location information directly within the Book class
- Using a high dimensional array to represent an entire bookcase or the library
- Implementing shelves as linked lists
- Using a fixed-size array for shelves
- Store location information externally
- Predefining initial locations for initial books

Managing the Data Structure

- Creating a dedicated service class to manage the data structure
- Extending an existing service class

- Storing the data structure in an unrelated class
- Implicitly managing locations within the Book class
- Using a list or hashmap to map books to their locations
- Storing location references within the Book class
- Storing book references within a Location class

Retrieving Information from the Data Structure

- Extracting location information implicitly through the structure itself
- Computing location details externally based on stored data
- Implementing distinct methods for each required retrieval function

Processing User Input

- Passing input parameters directly to another class
- Creating an object to encapsulate user input

Passing Information to the UI

- Implementing predefined TODO markers in the existing UI
- Developing an independent UI component for displaying book location data

Verifying User Input

- Checking whether a book already exists at a specified location
- Validating the feasibility of a location within the defined structure

5.4 Pilot Study

A pilot study was conducted with two participants to test the experiment execution process and gather early feedback on the task and the agent. This helped identify potential adjustments needed before the actual study.

One key finding was that the task appeared too complex, as neither participant was able to reach the final stages of completion. To address this, the task was slightly simplified. One layer of the location information hierarchy was removed, whereas location previously consisted of a region, bookcase and shelf, now it only consisted of bookcase and shelf. Additionally, some terminology within both the task description and the project itself was refined for better clarity. As such, UI classes were renamed from a “Tool” suffix to the more straightforward “Ui” suffix they have now. For the location hierarchy, “bookcase” and “shelf” were previously named “bookshelf” and “tier”.

The interview was also revised based on the pilot study. Some questions were found to be less relevant to the analysis, and a different order of questions proved more logical. As a result, the interview was refined to its current structure.

Technical issues with the agent were also observed, leading to the improvement of error messages within the plugin. Especially the hint to "Please try again" was added, which often solved the problem.
Additionally, an instruction was added to the explanation during the preparation phase to prevent situations where the agent might become unavailable during the experiment. Participants were told to reset the conversation or restart the IDE in these cases.

Regarding the agent's emotional expressions, one specific piece of feedback concerned the "concentrated" emotion, which was perceived as somewhat "evil". Consequently, its visual representation was adjusted to be a bit more appropriate, although this redesign was ultimately also negatively perceived at first sight.

Finally, feedback on the agent's proactive behavior indicated that proactive messages appearing while participants were still reading the task were especially distracting. To mitigate this, an initial five-minute delay for proactive messages was introduced, ensuring the agent would not interrupt the participant's initial understanding phase.

5.5 Data Collection

This section presents the data sources and the methods used for collection. In total, three primary sources of data were utilized:

Survey: Participants completed a survey consisting mainly of Likert scale questions to assess their experience with the agent.

Interview: A follow-up interview provided deeper insights into specific topics, allowing participants to elaborate on their survey responses and share additional thoughts.

Plugin/Source Code Data: During the task telemetry data, source code snapshots over time, and conversation logs with the agent were collected to analyze participant interactions in detail.

Survey

The survey serves as a quantitative data point within the study, providing insight into participants' subjective experiences and perceptions during the task. It primarily consists of 5-point Likert-scale items, which are used to comparably capture participants' subjective perceptions [61]. To account for nuances in responses, every section includes an optional text box where participants can elaborate on their answers if they feel additional explanation is necessary. An overview of the survey items and responses can be found in Appendix B.1.

The survey is divided into multiple sections, each corresponding to different aspects of the agent's functionality.

Before starting the task, participants complete the first section of the survey. This section collects basic demographic information and assesses their familiarity with key technologies used in the experiment. Specifically, participants rate their frequency of use with LLM-based tools, as well as familiarity with Java as a programming language and IntelliJ-based IDEs. Following this, an initial round of the Positive and Negative Affect Schedule (PANAS) is administered to establish participants' baseline emotional state [62].

After completing the task and uploading their results, participants proceed to the second and main part of the survey. This begins with a section evaluating the agent's perceived support in the different CPS phases, furthermore there are questions on divergent thinking, engagement with the task, and self-reliance.
A dedicated section follows for participants who interacted with the CPS Agent, focusing on their experiences with its proactive behavior.
The next section examines the emotional support provided by the agent. To quantify any emotional impact, a second round of PANAS questions is included, which later allows for comparison in the participant's emotional state before and after working with the agent.
The final section assesses the role perception of the agent and the participants' interaction with it.

Interview

The interview follows a semi-structured format, with a set of guiding questions while allowing flexibility for follow-up questions. If an answer was too vague or did not fully address the intended question, additional clarifying questions were posed.

The interview serves two primary purposes. The first part focuses on understanding how participants approached the task, including their problem-solving strategies and the extent of their divergent thinking. The second part allows participants to expand on topics covered in the survey, offering deeper insights into their experiences and perceptions.

The interview includes the following questions:

1. What problems could you identify in the task? In other words: what were concrete steps you needed to do to solve the task?
2. Were there problems you couldn't solve anymore?
3. Did you think about multiple different approaches to solve the problems of the task?
4. Do you think you worked creatively to reach your solution?
    a) Would you say the agent supported or hindered your creativity?

5. How did you feel throughout the task?
6. How would you describe the interaction with the agent?
7. Do you want to express any other thoughts?

For participants who preferred to conduct the interview in german, the questions were provided in translated form, see Appendix B.2.

The first question aims to assess how participants understand the task and how they structure their approach. Participants are encouraged to recall and articulate their solution ideas concretely. Follow-up questions were often used to elicit more details about participants' solutions.
The second question provides an opportunity for participants to discuss unresolved problems. If they mentioned any, they were asked to elaborate on their intended approaches.

Questions 3 and 4 explore divergent thinking and creativity. Question 3 specifically investigates whether participants considered multiple approaches to solve different aspects of task.
Question 4 shifts focus to self-perceived creativity, allowing participants to elaborate on their survey responses, as creativity might be difficult to quantify solely in a Likert-scale question. This question is also always followed up by a question about whether the agent supported or hindered the participant's creativity.

Question 5 addresses the emotional state of participants during the task, providing space for them to describe their feelings beyond what is captured in the PANAS assessment from the survey.
The sixth question invites participants to reflect on their interaction with the agent, offering further insights into their experience and perception of its role during the task.
Finally, the seventh question allows participants to share any additional thoughts, ensuring that any unaddressed concerns or observations could be recorded.

Plugin/Source Code Data

The plugin facilitates continuous data collection throughout the experiment, capturing both interactions within the agent and activity in the IDE. Three distinct types of data are recorded during the task: conversation logs with the agent, periodic snapshots of the source code, and telemetry data on code edits.

For conversation data, every interaction between the participant and the agent is logged with a timestamp, the message origin (either user or agent), and the message content. In the case of the CPS Agent, additional metadata is stored, including the agent's selected emotion, any pregenerated response options for the user, and whether the message was proactively generated. Furthermore, outputs from the summary agent, such as the conversation summary, identified boundaries, facts, goals, and challenges, are also logged.

To track code evolution, snapshots of all edited files are recorded approximately every minute, with an additional delay when the user is currently typing. This ensures that major changes to the project are captured without excessive data redundancy.

Edit telemetry data is also collected, logging each modification with a timestamp, the type of edit, and a fragment of the modified content. Edits are categorized into three types: `add`, when new characters are inserted, with the added characters stored as the fragment; `replace`, when existing characters are modified, storing the new characters as the fragment; and `remove`, when characters are deleted without replacement, logging the removed content.

At the end of the task, participants are instructed to upload their project folder, which contains the final state of the code along with all recorded data. While the plugin does not explicitly log this final state, it serves as the final reference point for analyzing participants' completed solutions.

5.6 Data Analysis

This section describes how the different sources of collected data were evaluated. Describing annotation procedures and statistical tests to enable a meaningful analysis.

Survey

As the survey primarily consists of quantitative Likert-style questions, minimal preprocessing is required. The optional notes participants could leave are treated as qualitative statements, and are analyzed in conjunction with responses to the interview questions, particularly interview questions 4 to 6 (Section 5.5).

For the PANAS ratings, the difference between the pre-task and post-task responses is calculated as the number of scale steps the answer increased or decreased. Although Likert scales are not strictly interval-based, this method offers an approximate measure of emotional change that may be induced by interaction with the agent.

All survey questions will be evaluated using the Mann–Whitney U statistic [63], a non-parametric test suitable for comparing ordinal Likert-scale data between two independent groups.

Interview

Two types of analyses are applied to the interview data. For interview questions 1 through 3, a structured annotation method is used. Questions 4 through 6 are evaluated using qualitative and descriptive methods. Answers to question 7, similar to the additional survey notes, are evaluated qualitatively and mentioned in conjunction with previous questions.

The annotation procedure for questions 1 to 3 begins by segmenting each participant's response into individual statements at the sentence level. Each of these statements is then analyzed for

the presence of either solution ideas or subproblems, as defined in Section 5.3. Depending on the specific interview question, the identified solution ideas are categorized differently: for Question 1, they are interpreted as implemented strategies; for question 2, they refer to approaches that were considered for unsolved subproblems; and for question 3, they represent alternative strategies that the participant contemplated but did not pursue. In situations where participants gave answers to subsequent questions in advance, statements are reassigned to the according question.
Subproblems mentioned in question 1 are recorded as identified subproblems by the participant for the task. Subproblems are also considered as identified when a respective solution idea is mentioned.

Since the described annotation of solution ideas and subproblems is binary per participant, the Fisher's Exact Test [64] is used to evaluate statistical differences between the two conditions.

For interview questions 4–6, a BERT-based model for german sentiment analysis [65] as well as VADER based sentiment analysis [66] were tried, but the resulting scores ultimately deemed too inaccurate. Instead, interview questions 4–6 are analyzed solely qualitatively along with interview question 7 and the additional survey notes.

Plugin and Project Data

The plugin and final project state deliver multiple types of data which are annotated and analyzed as follows:

Each conversation between the agent and user is annotated with several aspects in mind. First, all messages are examined for mentioned solution ideas and subproblems using the same annotation procedure as described for the interview data. Since this data relates to the annotation data from the interview, it will later be analyzed in conjunction with it.
In addition, each message is assigned a CPS phase, which is initially inferred through a pass using the GPT-4o-mini model, incorporating the CPS Agent's role system prompt. These labels are then manually reviewed and corrected where necessary to ensure accuracy.
Each message is assigned to the most advanced CPS phase that its content indicates. For instance, if a message contains a proposed idea, discusses its advantages and disadvantages, and includes corresponding implementation details, it is classified under the *Implement* phase. Although such a message may also be relevant to the *Idea* or *Develop* phases, the presence of implementation details signifies that the progression toward a concrete solution has already been completed, thus not invoking further engagement for the *Idea* or *Develop* phases.
The focus for this data lies on the distribution balance across the different CPS phases, which is quantified by calculating the Shannon entropy [67].

Furthermore, agent messages are annotated to capture instances of positive reinforcement directed at the user, while user messages are reviewed to determine whether they involve direct requests for code or problem solutions from the agent.

The telemetry data is analyzed to investigate interaction behavior in more detail. Specifically, code segments that were potentially copied from the agent are identified by checking for edits that contain at least one full line of code from a previous agent response, appear chronologically after that response, and are of type `add` or `replace`. Additionally, the frequency of each edit type, `add`, `replace`, and `remove`, is calculated for each participant.
Finally, the overall number of lines of code inserted to the project during the task is computed using `git diff`.

Statistical differences in these behavior metrics are evaluated using t-tests [68]. Throughout the entire analysis, all statistical tests are employed in their two-sided variants to identify statistically significant differences between conditions, with trends subsequently interpreted based on the resulting test statistics.

The final state of the project is annotated to determine which subproblems are addressed in the code and which solution ideas were implemented. A solution idea does not need to be fully working to be counted, as the focus is on design decisions rather than complete implementations. This also results in annotation data similar to that of the interview and conversation, and will therefore be analyzed alongside it as well.

All the different annotations of solution ideas and subproblems enables various Fisher's Exact Tests as well as t-tests [64, 68] to be performed, comparing design tendencies across groups. By triangulating across the different forms of data, a more holistic view of each participant's approach and experience can be constructed.

6 Results

In this chapter, we present the collected data from our study along with corresponding interpretations. We begin by examining the participant pool, with particular focus on their familiarity with the technologies employed in the experiment.

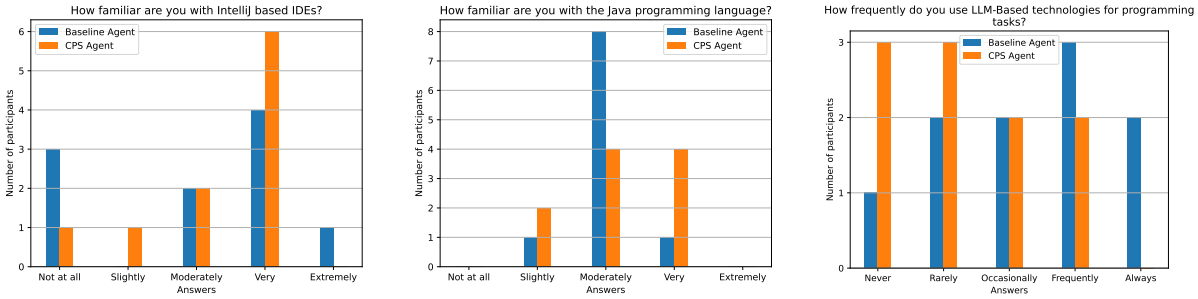
Subsequently, we address the hypotheses outlined in Section 5.1. Each hypothesis is handled in a dedicated section, where we first provide a high-level overview of the findings, followed by a description of relevant data and a more detailed discussion in light of the respective hypothesis.

6.1 Participants

Participants were required to meet the following prerequisites: a minimum of one year of programming experience, a basic understanding of the Java programming language, and the ability to comprehend tasks written in english.

On average, participants in the baseline group reported 6.5 years of experience, while those in the CPS Agent group reported an average of 8.0 years. In addition, Figure 6.1 illustrates the participants' self-reported familiarity with the core technologies used in the experiment. While participants in the CPS Agent group reported slightly lower familiarity with LLM-based technologies, this difference resulted from the randomized nature of the a priori group assignment. Although such imbalance may introduce potential bias, the difference was not found to be statistically significant and is therefore acknowledged as a possible, but limited, confounding factor.

Participant 6 did not fully meet the predefined selection criteria, as they reported having no prior experience with the Java programming language. Consequently, this participant was excluded from the final analysis to maintain consistency and validity across the groups. Their results and responses are briefly addressed in Chapter 7. The resulting dataset comprises 10 participants in each experimental group.



(a) IntelliJ based IDEs (b) Java (c) LLM-Based technologies

Figure 6.1: Participants' familiarity with the technologies used in the experiment.

6.2 RQ1: Problem-Solving

For RQ1, we take a step back to assess individual task performance from a higher-level perspective. This includes evaluating general task completion and participants' comprehension of both the task and their own solutions.

6.2.1 H1: The CPS Agent increases the extent to which participants solve the task

We begin our evaluation with a more high-level analysis of task completion. To investigate H1, this section examines the extent to which participants developed approaches for the different subproblems, either through actual code implementation or through stated approaches discussed during the interview. Our findings reveal no significant differences between the experimental groups, suggesting that task completion was primarily driven by individual participant performance rather than experimental condition.

Data

It is important to note that, both in this section and throughout the following analyses, a complete and functional code implementation was not required in our annotation. The key criterion is whether the participant demonstrated a concrete approach to solving the respective subproblem, indicating a meaningful engagement with the task.

Table 6.1 presents the number of participants who partially implemented code solutions for the individual subproblems of the task. Additionally, it shows how many participants indicated during the interview that they encountered problems they could not fully solve within the timeframe of the experiment but had considered potential solutions or outlined an approach.

Subproblem	Baseline Agent	CPS Agent
Creating a Data Structure	10 (+ 0)	10 (+ 0)
Managing the Data Structure	10 (+ 0)	10 (+ 0)
Retrieving Information from the Data Structure	7 (+ 1)	10 (+ 0)
Processing User Input	6 (+ 2)	7 (+ 0)
Passing Information to the UI	6 (+ 2)	8 (+ 1)
Verifying User Input	1 (+ 1)	2 (+ 2)

Table 6.1: Subproblems that participants implemented code for (+ stated they had an approach for)

Discussion

The results do not indicate a clear or significant difference between the two agent conditions in terms of implemented solutions and approaches to unsolved subproblems. Given that both agents supported the task and were ultimately used in similar ways, it is reasonable that performance remains largely dependent on the individual capabilities of the participants rather than the specific agent.

One notable observation is that participants working with the CPS Agent slightly more frequently addressed the verification of user input. This could be attributed to the CPS Agent's proactive messages, which occasionally highlighted this aspect of the task, unlike the baseline agent, which more rarely brought it up independently.

6.2.2 H2: The CPS Agent facilitates a deeper understanding of the problem in participants

To investigate H2, we examine participants' underlying understanding of the task and their own solutions. This includes analyzing how thoroughly participants recalled the subproblems of the task and their corresponding approaches.
Participants across both conditions demonstrated similar recall of their solution strategies and comparable task comprehension during the interviews. Participants in the baseline group reported a greater enhancement of their understanding by the agent during the survey, while participants in the CPS Agent group recalled one specific aspect of their solutions more frequently. However, overall differences were marginal, suggesting that the CPS Agent had limited impact on participants' conceptual understanding of the problem.

Data

First, we examine how many distinct subproblems of the task were identified by participants during the interview. This analysis is based on responses to interview question 1, as shown in

Subproblem	Baseline Agent		CPS Agent	
	Interview	Conversation	Interview	Conversation
Creating a Data Structure	10	10	10	10
Managing the Data Structure	9	8	8	9
Retrieving Information from the Data Structure	7	9	7	9
Processing User Input	1	4	5	3
Passing Information to the UI	5	7	8	8
Verifying User Input	2	3	2	5

Table 6.2: Subproblems that participants identified in the interview (first value) and that are mentioned in the conversation history with the agent (second value).

Table 6.2. To explore the role of the agent in the identification of subproblems, we also assess which subproblems were mentioned during the agent-participant conversation, which can also be seen in Table 6.2.

Additionally, we analyze the solution ideas recalled by participants as captured in interview question 1. These results are summarized in Table 6.3. For comparison, we also report the number of solution ideas that were actually implemented in code, allowing us to contrast the extent of participants' mental model with their realized solutions.

In addition, Figure 6.2 presents survey questions related to H2. These questions provide further insight into how participants perceived their understanding of the task. While generally most questions in the survey followed a 5-point Likert scale ranging from "strongly disagree" to "strongly agree", some questions follow a different scale, such questions are marked with a different color scheme in the visualizations.

Discussion

The interview data reveals generally similar patterns between the two groups. When examining the subproblems identified by participants, both groups recognized most issues at comparable rates.

One exception that could be noted is the subproblem "Processing the user input", which was recalled explicitly by only one participant in the baseline group, but by half of the participants in the CPS Agent group, though statistically non-significant according to fisher's exact test ($p = 0.1409$). A possible explanation from a more qualitative standpoint could be that participants often summarized "Processing the user input" and "Passing Information to the UI" as more generally adapting the user interface, not differentiating the specifics of handling input and output.

The subproblems mentioned during interactions with the agents also align closely between the two conditions.

Solution idea	Baseline Agent	CPS Agent
Creating a Data Structure		
A Location class storing book location details	6 / 7	5 / 5
Multiple classes representing the hierarchy of book locations	6 / 6	5 / 6
Using a recursive structure	0 / 0	0 / 0
Using inheritance to create the hierarchy of book locations	0 / 0	0 / 0
Storing location information directly within the Book class	1 / 1	0 / 2
Using a high dimensional array	1 / 1	0 / 0
Implementing shelves as linked lists	0 / 0	0 / 0
Using a fixed-size array for shelves	0 / 0	0 / 0
Store location information externally	1 / 1	0 / 0
Predefining initial locations for initial books	3 / 6	4 / 7
Managing the Data Structure		
Creating a dedicated service class to manage the data structure	1 / 2	1 / 2
Extending an existing service class	5 / 5	3 / 5
Storing the data structure in an unrelated class	0 / 2	0 / 2
Implicitly managing locations within the Book class	4 / 4	3 / 3
Using a list or hashmap to map books to their locations	0 / 3	4 / 4
Storing location references within the Book class	3 / 5	4 / 5
Storing book references within a Location class	0 / 1	1 / 3
Retrieving Information from the Data Structure		
Extracting location information implicitly	0 / 4	0 / 5
Computing location details externally based on stored data	3 / 3	2 / 6
Implementing distinct methods for each required retrieval	1 / 4	2 / 5
Processing User Input		
Passing input parameters directly to another class	0 / 3	0 / 1
Creating an object to encapsulate user input	0 / 3	1 / 5
Passing Information to the UI		
Implementing predefined TODO markers in the existing UI	2 / 6	2 / 7
Developing an independent UI component	0 / 0	1 / 1
Verifying User Input		
Checking whether a book already exists at a specified location	1 / 1	1 / 1
Validating the feasibility of a location within the defined structure	1 / 1	0 / 1

Table 6.3: Recalled solution ideas of participants' solutions / Implemented solution ideas in participants' code

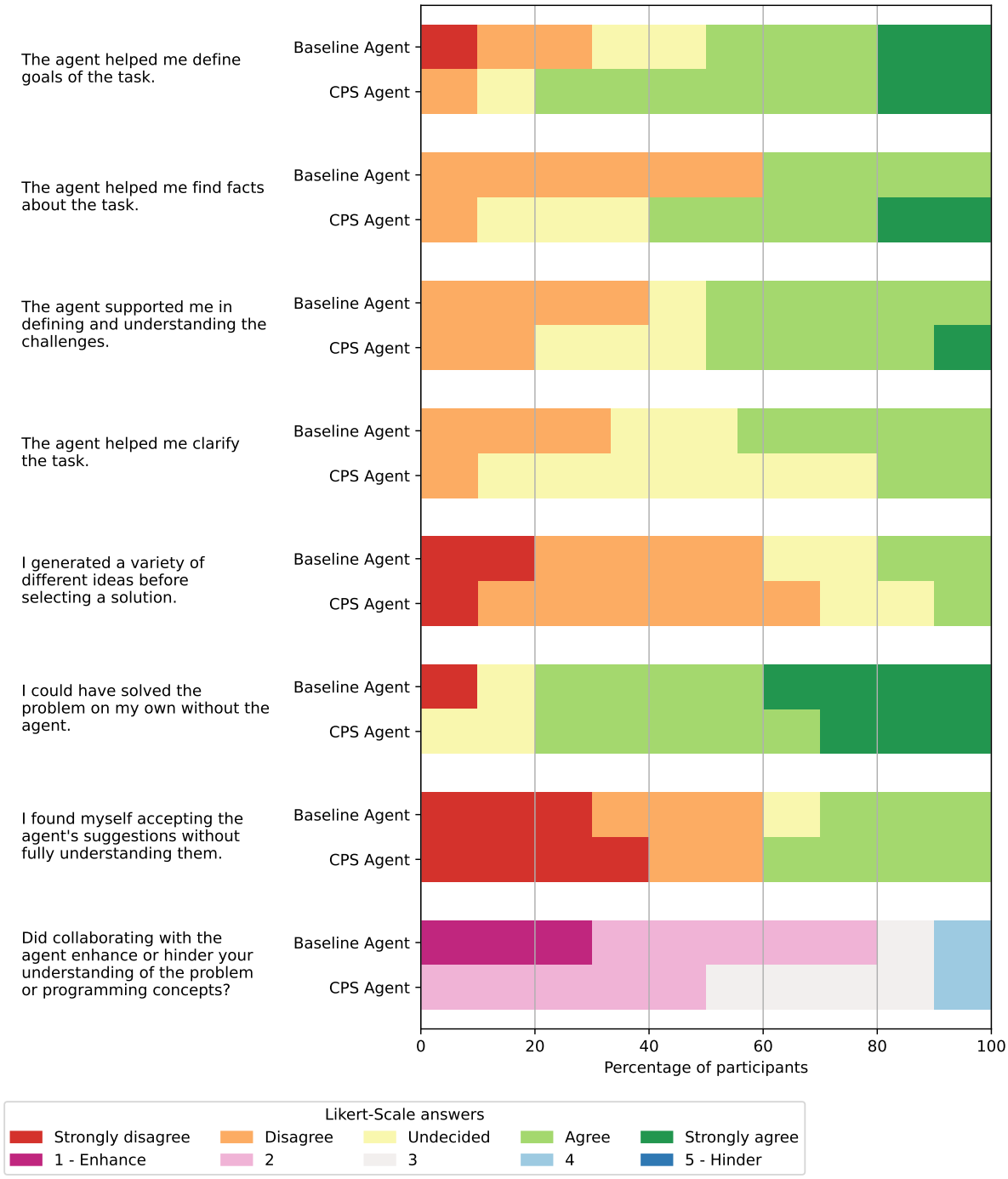


Figure 6.2: Survey questions regarding H2

Analysis of the recalled and implemented solution ideas yields a few points of interest. Participants in the CPS Agent group more frequently described explicit strategies for associating books with their locations. Across both groups, few participants articulated detailed strategies for retrieving information, particularly when retrieval was handled implicitly by the structure itself. Similarly, only a small number went into detail about processing user input.

Survey responses reinforce the general similarity between the two groups, although this was somewhat unexpected. The hypothesis had anticipated that collaboration with the CPS Agent would promote deeper exploration of the task. Instead, the CPS Agent often guided participants quickly to subsequent phases in the CPS, which may have limited reflective thinking.
The survey item "The agent helped me find facts about the task" showed a slight positive effect ($p = 0.0716$) according to the Mann-Whitney U test. Conversely, the final question "Did collaborating with the agent enhance or hinder your understanding of the problem or programming concepts?" revealed a slight negative trend ($p = 0.0942$), where participants in the baseline group reported a stronger perceived enhancement of their understanding. This may be due to the baseline agent providing more comprehensive and explanatory messages when receiving the full task description at once, whereas the CPS Agent typically offered brief overviews and then prompted participants to decide what problem to address first.

6.3 RQ2: Effectiveness on Creative Problem-Solving Support

For RQ2, we continue exploring problem-solving, but with a specific focus on how the CPS Agent influenced participants within the framework of the CPS. This includes examining perceived support for the distinct phases of the CPS, creativity in the form of divergent thinking, and the extent of participants' self-reliance.

6.3.1 H3: The CPS Agent provides stronger support for the creative problem-solving process

To evaluate H3, we analyze a range of data sources that reflect how effectively the agent supports the different phases of the CPS. Survey responses and annotation data consistently indicate that the CPS Agent enabled more fine-grained support compared to the baseline condition. However, this increased alignment with the CPS came at the expense of a reduced perceived support during the *Implementation* phase.

Data

We begin with an overview of the interaction volume in terms of exchanged messages between participants and the agent, shown in Table 6.4. For additional context we also provide the length

Metric	Baseline Agent	CPS Agent
Mean number of messages from agent	13.50	42.80
Median number of messages from agent	12.00	44.00
Mean number of messages from user	13.70	32.50
Median number of messages from user	12.00	33.50
Mean number of text messages by user	13.70	14.70
Median number of text messages by user	12.00	14.00
Mean number of pregenerated messages used by user	-	17.80
Mean number of proactive agent messages	-	10.30
Mean message length in characters of agent messages	2624.04	489.40
Median message length in characters of agent messages	2793.00	379.00

Table 6.4: General usage metrics of the agent

of agent responses, the amount of proactive messages from the agent, as well as the number of pregenerated and written messages sent by the participant.

A further aspect lies in the categorization of conversation messages into the different CPS phases. Table 6.5 presents several metrics related to these phases, including their overall distribution across conversations and the proportion of participants whose interaction histories contained each CPS phase. To quantify the diversity of CPS phase engagement, we compute the Shannon entropy [67] of the phase distribution. A higher entropy value indicates a more balanced engagement across the various phases of the CPS process.

Complementing the conversational analysis, survey responses regarding perceived CPS support are visualized in Figure 6.3. These results further illuminate how participants evaluated the agent's support across different aspects of the CPS.

Discussion

As anticipated, a substantial difference in the number and length of exchanged messages is evident between the two conditions (Table 6.4). This is primarily attributed to the design of the CPS Agent, which promotes more extended interaction dialogues. The inclusion of proactive messages and pregenerated response options additionally led to a higher message count. Interestingly, we can observe a similar interaction volume with regard to written text messages, which might suggest that participants felt the need to initiate with the agent at comparable rates.

Turning to the CPS phases (Table 6.5), one of the CPS Agent's objectives was to increase engagement in phases other than *Implement*, with the intent of fostering deeper problem understanding prior to code production. The results indicate that, although *Implement* remains the dominant phase, the CPS Agent significantly increased interaction dedicated solely to the *Clarify*, *Idea*,

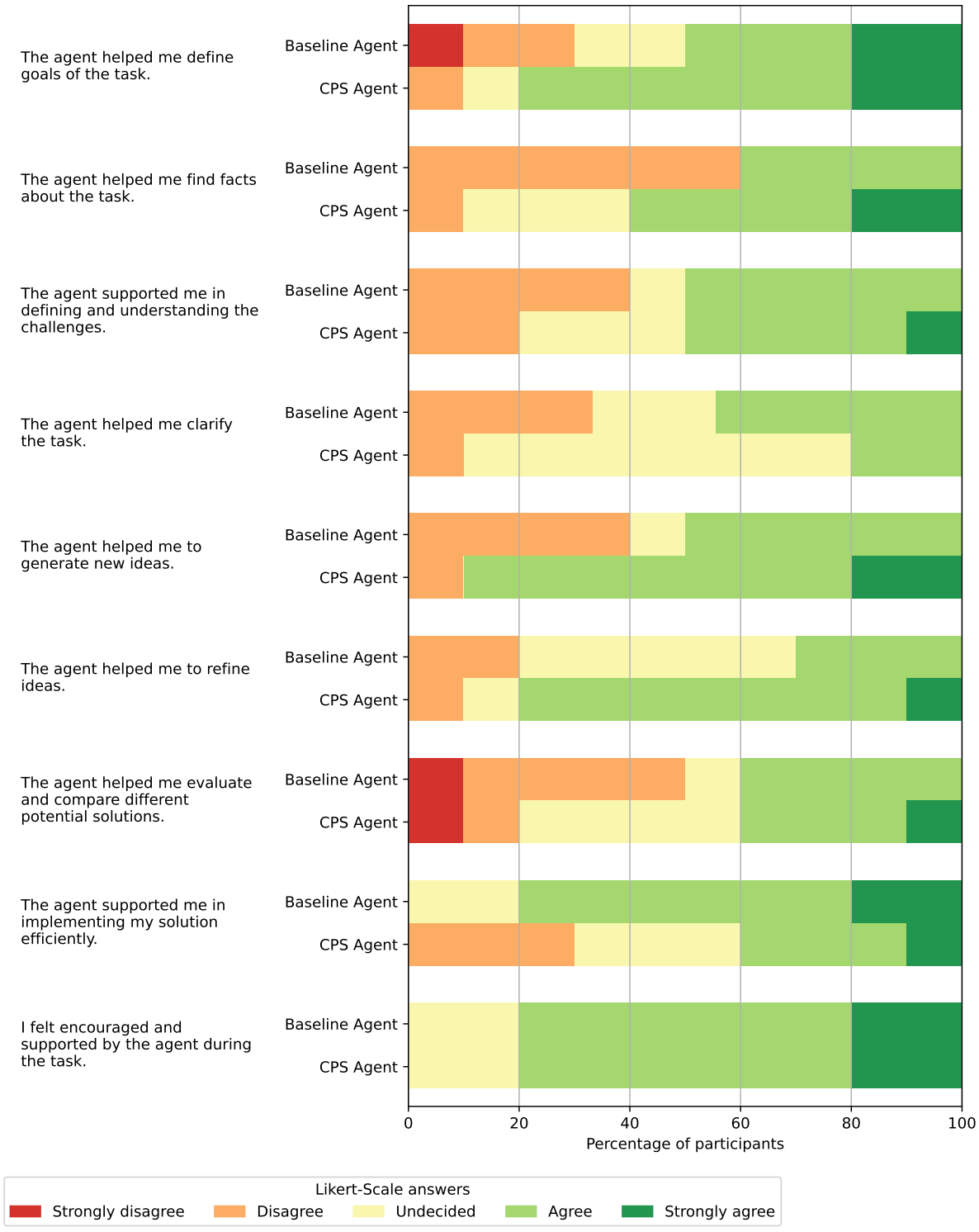


Figure 6.3: Survey questions regarding H3

Metric	Baseline Agent	CPS Agent
Mean percentage of messages in *Clarify* phase	11.31%	14.34%
Mean percentage of messages in *Idea* phase	2.78%	12.47%
Mean percentage of messages in *Develop* phase	5.89%	12.25%
Mean percentage of messages in *Implement* phase	80.02%	60.94%
Mean entropy of distribution of phases	0.57	1.06
Percentage of participants with a message in *Clarify* phase in conversation	90.00%	100.00%
Percentage of participants with a message in *Idea* phase in conversation	50.00%	100.00%
Percentage of participants with a message in *Develop* phase in conversation	50.00%	100.00%
Percentage of participants with a message in *Implement* phase in conversation	100.00%	100.00%

Table 6.5: Metrics regarding the distribution of CPS phases in participants' conversations with the agent

and *Develop* phases. This shift is reflected in a significantly higher entropy of the message phase distribution compared to the baseline condition ($p = 0.0002$). This may suggest that the CPS Agent was at least partially successful in encouraging more deliberate and reflective problem-solving behavior.

In line with prior findings [29], many participants in the baseline group tended to skip directly from clarifying the problem to implementation, omitting intermediate cognitive steps that could support creativity. We observe half of the baseline participants had no interaction turns categorized as belonging solely to the *Idea* or *Develop* phases. In contrast, for all participants interacting with the CPS Agent we observed some dedicated engagement across all CPS phases.

Survey responses further support these observations. The statement "The agent helped me to generate new ideas" showed a significant difference ($p = 0.0386$), indicating a positive effect of the CPS Agent on ideation. A similar trend was observed for the item "The agent helped me to refine ideas" ($p = 0.0441$). Furthermore, survey item "The agent helped me find facts about the task." exhibits a slight positive trend ($p = 0.0716$). This might suggest that the CPS Agent supported participants meaningfully during the early *Clarify* and *Idea* phases.

Conversely, a negative effect ($p = 0.0712$) was found for the item "The agent supported me in implementing my solution efficiently", which indicates that some participants may have experienced the extended dialogic structure of the CPS Agent as a hindrance to swift implementation.

6.3.2 H4: The CPS Agent fosters increased divergent thinking

H4 focuses on participants' creativity, examined through the concept of divergent thinking within the CPS. Divergent thinking is particularly relevant during the *Idea* phase, where multiple alternative solutions are generated. To assess this, we consider the number and variety of solution ideas produced by participants, relevant survey responses regarding ideation and divergent thinking, as well as relevant interview questions. While both groups generated a similar number of ideas, participants interacting with the CPS Agent reported a higher perceived support for ideation. Nonetheless, the overall level of divergent thinking remained limited across both conditions, suggesting that the CPS Agent, despite its design intention, did not meaningfully support broader creative exploration.

Data

To evaluate divergent thinking, we compile all identifiable solution ideas generated by participants, as shown in Table 6.6. These ideas were gathered through multiple sources: the interview, the conversation with the agent, and the implemented code within the project. For the interview, the first three questions are considered, as they specifically target problem identification and alternative approaches. Within the conversation data, solution ideas are annotated for both the participant and the agent. While the agent's suggestions may not always have been consciously acknowledged or utilized by the participant, they are included here under the assumption that they contributed to the participant's ideation process. As for the source code, all implemented changes are considered as manifestations of solution ideas, irrespective of their level of completion.

To complement this qualitative annotation, relevant questions from the survey are presented in Figure 6.4, which provide a quantitative lens to the extent of participants' self-reported divergent thinking and ideation during the task.

Furthermore, we take a look at responses to interview question 4, which explicitly inquires participants whether they perceived themselves to be working creatively. This question was always followed by a follow-up (Item 4a), questioning whether participants felt the agent enhanced or hindered their creativity. The follow-up responses emphasize the perceived effect of the agent on creativity.

Solution idea	Baseline Agent	CPS Agent
Creating a Data Structure		
A Location class storing book location details	7	7
Multiple classes representing the hierarchy of book locations	8	6
Using a recursive structure	0	1
Using inheritance to create the hierarchy of book locations	0	1
Storing location information directly within the Book class	1	3
Using a high dimensional array	3	1
Implementing shelves as linked lists	0	1
Using a fixed-size array for shelves	0	0
Store location information externally	1	0
Predefining initial locations for initial books	7	9
Managing the Data Structure		
Creating a dedicated service class to manage the data structure	5	3
Extending an existing service class	8	6
Storing the data structure in an unrelated class	3	3
Implicitly managing locations within the Book class	5	5
Using a list or hashmap to map books to their locations	5	5
Storing location references within the Book class	5	8
Storing book references within a Location class	1	4
Retrieving Information from the Data Structure		
Extracting location information implicitly	7	7
Computing location details externally based on stored data	5	6
Implementing distinct methods for each required retrieval	6	6
Processing User Input		
Passing input parameters directly to another class	4	2
Creating an object to encapsulate user input	4	5
Passing Information to the UI		
Implementing predefined TODO markers in the existing UI	8	8
Developing an independent UI component	1	1
Verifying User Input		
Checking whether a book already exists at a specified location	3	4
Validating the feasibility of a location within the defined structure	2	5

Table 6.6: All the solution ideas mentioned in either interview or conversation, as well as implemented in the code

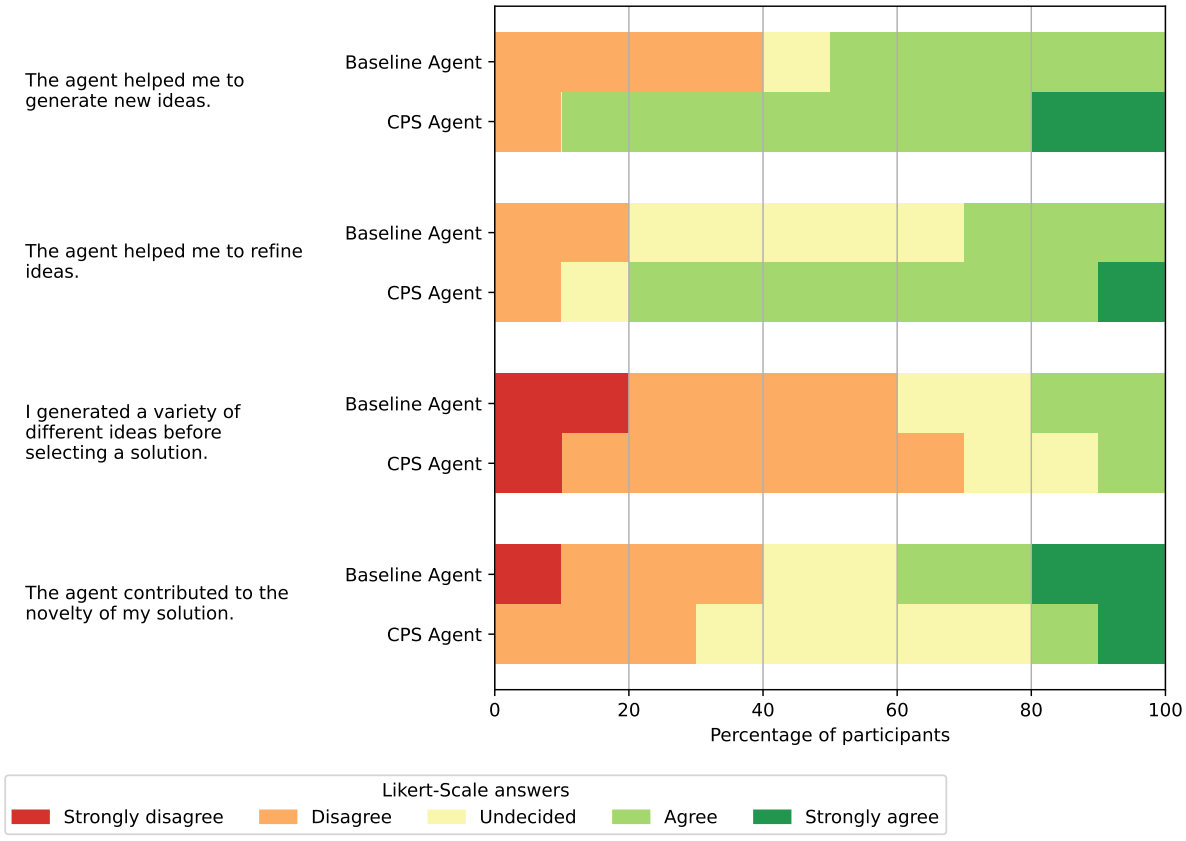


Figure 6.4: Survey questions regarding H4

Discussion

The analysis of solution ideas generated across both conditions reveals no substantial difference in the degree of divergent thinking between the baseline and CPS Agent groups. The overall distribution of unique ideas, as detailed in Table 6.6, is relatively balanced, and while some specific concepts appear more frequently in one group than the other, no consistent trend emerges in favor of either condition. While this results in an inconclusive analysis, this might indicate a drawback of the programming task itself, which while offering a variety of opportunities for design decisions, is ultimately limited in feasible solution ideas given the task and limited timeframe.

Survey responses (Figure 6.4) suggest a more positive perception of support for idea generation and refinement in the CPS Agent condition, as already discussed in Section 6.3.1. However, the item “I generated a variety of different ideas before selecting a solution” received generally low agreement scores across both groups, indicating that participants from neither group engaged very consciously in divergent thinking. The time limit appeared to play a significant role, as several participants reported in the interview that they prioritized task completion over exploring potentially better alternatives.
Furthermore, this pattern might again highlight the CPS Agent’s tendency to prematurely advance the participant through the CPS phases. Had the agent more effectively sustained

engagement within the *Idea* phase, a broader range of ideas and a stronger perception of support for divergent thinking might have emerged.

Responses to interview question 4 revealed a mixed set of perspectives. Participants' reflections on working creatively appeared to be influenced less by the specifics of the task and more by their general understanding of what constitutes creative work in programming contexts.

In the baseline group, participants frequently characterized the agent as a tool to realize their ideas rather than a collaborator, indicating that they largely excluded it from their creative process. Several participants additionally reported that they relied on the agent primarily for initial planning or implementation assistance, following its plan without much further engagement.

In contrast, participants in the CPS Agent group expressed a broader range of experiences. Several participants acknowledged the agent's attempt to take on a more collaborative role of a pair programming partner. While this approach was welcomed by certain participants, others found it misaligned with their expectations, preferring instead a tool-like interaction.
Multiple participants in the CPS Agent group reported that the agent had hindered their creativity. These participants often described a process in which they collaboratively generated an initial idea with the agent and subsequently adopted this first suggestion, because of time pressure and initial satisfaction with the proposal. This then ultimately led to the perception that this had limited their own creative contribution to the task.

6.3.3 H5: The CPS Agent enhances participants' self-reliance

H5 investigates the extent to which the CPS Agent fosters greater self-reliance in participants during the task. To address this hypothesis, we analyze both behavioral telemetry data and self-reported perceptions gathered through the survey. Our findings indicate that participants in the CPS Agent condition demonstrated a higher degree of self-reliance in code production, as evidenced by increased manual coding activity and reduced perceived dependence on agent-generated code. In contrast, the baseline condition showed a notable correlation between reliance on the agent for problem-solving and reduced code editing activity. Despite this behavioral trend, participants in the baseline group still reported an overall strong sense of personal agency in problem-solving, aligning with findings from prior studies that highlight a gap between perceived and actual reliance on GenAI tools [2, 5].

Data

The telemetry data provides insights into the interaction patterns between participants and the agent. An overview of relevant metrics and annotations is presented in Table 6.7. As part of the annotation process, user messages were examined for instances in which participants explicitly requested solutions or code generation from the agent. The table reports both the average number and the proportion of such messages relative to the total number of user messages. In

addition to annotated message behavior, multiple quantitative telemetry indicators are considered. These include the total number of code fragments generated by the agent and the number of those fragments that were subsequently copied into the participant's code. Furthermore, we examine final code insertions based on a `git diff` analysis of the project state. The collected edit telemetry from the plugin is also analyzed, capturing the amount of `add`, `replace`, and `remove` operations performed in the IDE throughout the task.

Complementing the behavioral data, survey responses related to perceived self-reliance are analyzed. These questions, represented in Figure 6.5, specifically assess the degree to which participants felt they relied on the agent for code generation and problem-solving support. Additional survey questions probe participants' engagement with the task and their subjective experience of flow.

Metric	Baseline Agent	CPS Agent
Mean number of messages asking for code or a solution	8.00	9.40
Mean percentage of messages asking for code or a solution	65.13%	27.23%
Mean number of code fragments generated by the agent	21.10	11.30
Mean number of code fragments copied from	9.11	7.00
Mean percentage of code fragments copied from	55.37%	63.63%
Mean number of code line insertions	193.30	161.00
Median number of code line insertions	195.50	154.50
Mean number of `add` edits	567.78	1021.11
Median number of `add` edits	324.00	1142.00
Mean number of `replace` edits	14.56	15.00
Median number of `replace` edits	14.00	10.00
Mean number of `remove` edits	119.78	211.00
Median number of `remove` edits	107.00	218.00

Table 6.7: Telemetry data and amount of copied code fragments

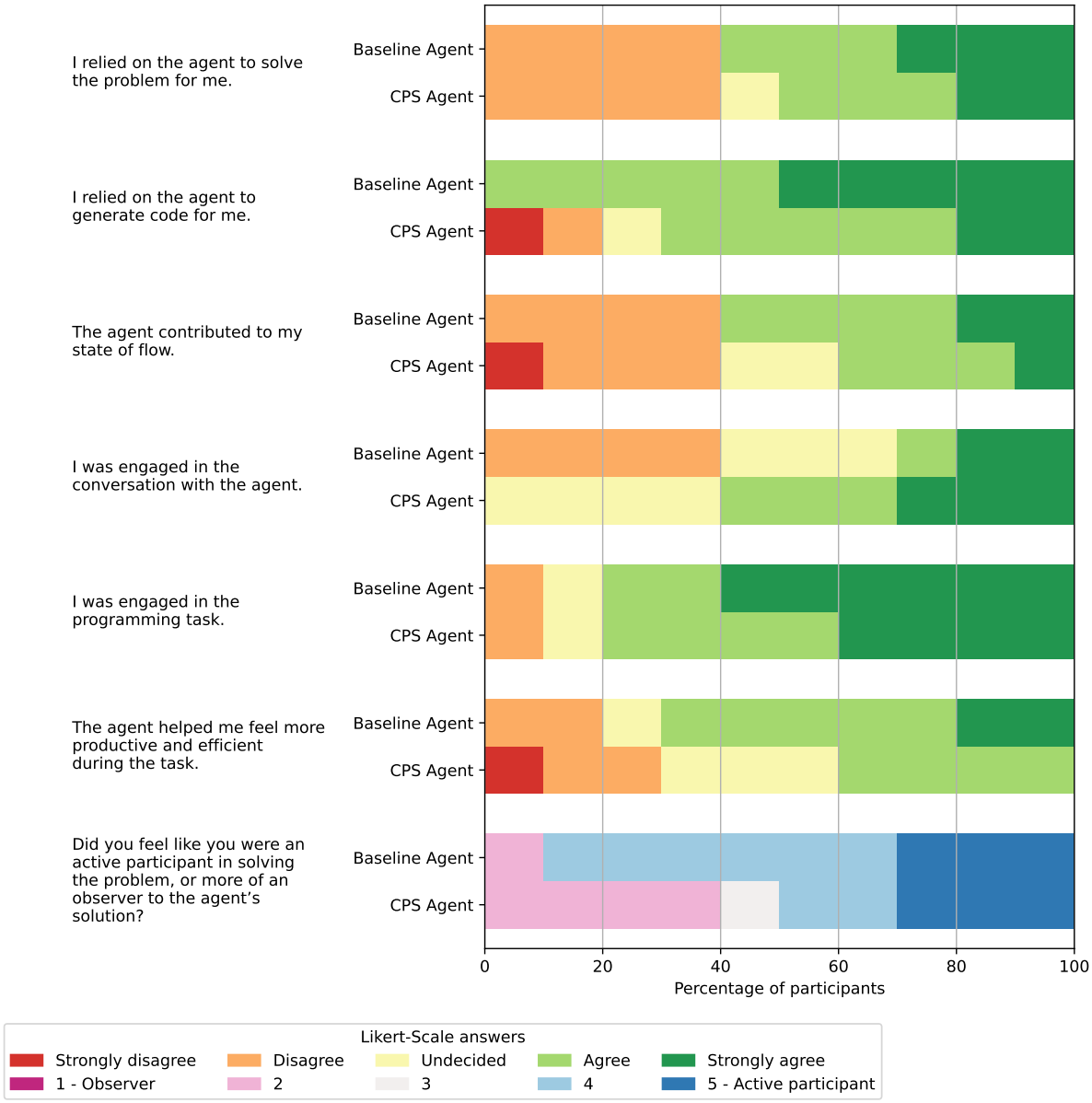


Figure 6.5: Survey questions regarding H5

Discussion

While participants interacting with the CPS Agent issued a slightly greater absolute number of requests for code or solutions, these accounted for a substantially smaller proportion of their overall messages (27.23%) compared to those using the baseline agent (65.13%). This might suggest that participants in the CPS Agent group engaged with the agent in a more diversified manner, rather than relying primarily on it for code generation.

Regarding code generation behavior, the baseline agent produced nearly twice as many code fragments on average as the CPS Agent ($p = 0.0460$). Despite this, participants copied code from a smaller proportion of these fragments (55.37%) compared to those from the CPS Agent

(63.63%). One explanation for the greater amount of code fragments in the baseline group was the generation of additional examples, which seemed redundant in some cases.

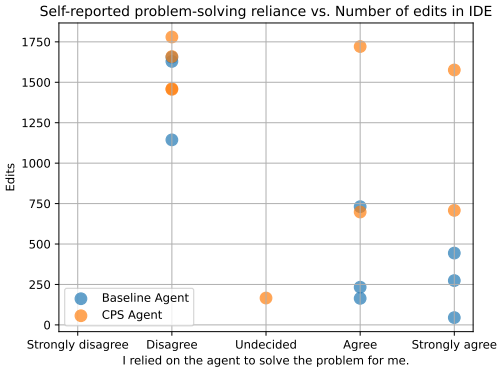
Edit activity within the IDE also reveals important differences. Participants in the CPS Agent group performed a higher number of `add` ($p = 0.0821$) and `remove` ($p = 0.0962$) operations, which may indicate more hands-on code development rather than relying heavily on agent-generated content. This observation supports the hypothesis that the CPS Agent fosters greater self-reliance by promoting user-initiated code production.

The number of final code line insertions was marginally lower for the CPS Agent group, although not significant, potentially reflecting less boilerplate or more concise code.

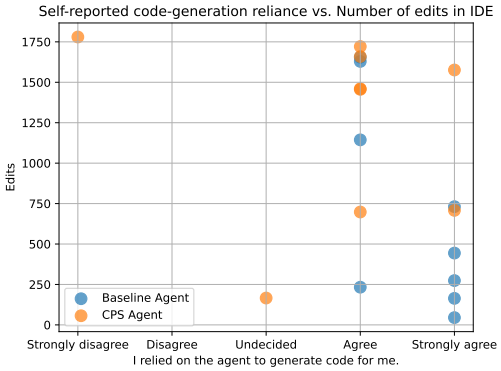
Survey results allow for further interpretation. In response to the statement "I relied on the agent to generate code for me", all participants in the baseline group agreed, whereas some in the CPS Agent group disagreed ($p = 0.0686$). This disparity suggests a trend toward reduced dependency on the agent for code production in the CPS Agent condition. Additionally, the item "I was engaged in the conversation with the agent" showed a slight increase in engagement for the CPS Agent group ($p = 0.0993$), aligning with the agent's design goal of fostering more interactive problem-solving.

To examine the relationship between participants' self-reported reliance on the agent and their actual level of engagement with code generation, we plotted responses to the interview items related to reliance on the agent for problem-solving and code generation against the number of code edits performed during the task (Figure 6.6).
A particularly strong negative correlation was observed between reliance on the agent for problem-solving and the number of performed edits within the baseline group, as indicated by Pearson's correlation coefficient [69] ($r = -0.90$, $p = 0.0009$). While this relationship was not statistically significant within the CPS Agent group, the overall correlation across both groups remained significant ($r = -0.63$, $p = 0.0052$), suggesting that increased reliance on the agent was generally associated with reduced individual engagement in code production.
A similar, though less pronounced, trend was found for reliance on the agent for code generation. Participants in both groups reported relatively high reliance levels, yet a moderate negative correlation with the number of edits was still evident in the combined dataset ($r = -0.41$, $p = 0.0877$), which again was more pronounced within the baseline group ($r = -0.70$, $p = 0.0358$).

The final survey item from Figure 6.5, "Did you feel like you were an active participant in solving the problem, or more of an observer to the agent's solution?", did not yield a significant difference between groups ($p = 0.2991$), contrary to expectations. Interestingly, a slight, though non-significant, shift towards a passive role was observed in the CPS Agent group. This also aligns with the prior observations about creativity (Section 6.3.2), where participants reported that the guidance of the agent through the CPS phases led to the perception of a reduced impact of their own contribution. Although the CPS Agent was designed to follow the CPS as a



(a) Reliance for problem-solving



(b) Reliance for code generation

Figure 6.6: Relationship between participants' self-reported reliance on the agent for problem-solving and code generation and the number of code edits performed during the task.

dialogue structure, its tendency to advance the conversation to subsequent CPS phases may have inadvertently limited opportunities for participants to engage more deeply within a given phase. These findings additionally may reflect the emerging concerns regarding overreliance on generative AI in problem-solving contexts [2, 5]. As we saw in Figure 6.6, participants in the baseline group frequently reported substantial reliance on the agent for both problem-solving and code generation. Despite this, they predominantly described themselves as playing an active role in the problem-solving process (Figure 6.5). In contrast, more participants in the CPS Agent group characterized their role as more passive, which may suggest a greater awareness of the agent's influence and involvement in the cognitive aspects of the task.

6.4 RQ3: Perception of the Agent

For RQ3, we shift focus to the participants' general perception of the agent and its personality. This includes examining how the agent's role was perceived, how its proactive behavior and emotional intelligence were received, and the degree of affective connection participants developed during the interaction.

6.4.1 H6: The CPS Agent's role is perceived as more social

H6 explores the extent to which participants perceived the CPS Agent as adopting a more socially-oriented role during the task. This hypothesis is inspired by prior work [27], which also assessed role perception in human-agent interaction within programming contexts. To examine this, we incorporated a replication of their role perception assessment into our survey. Our results show that participants in the CPS Agent condition more frequently attributed social roles

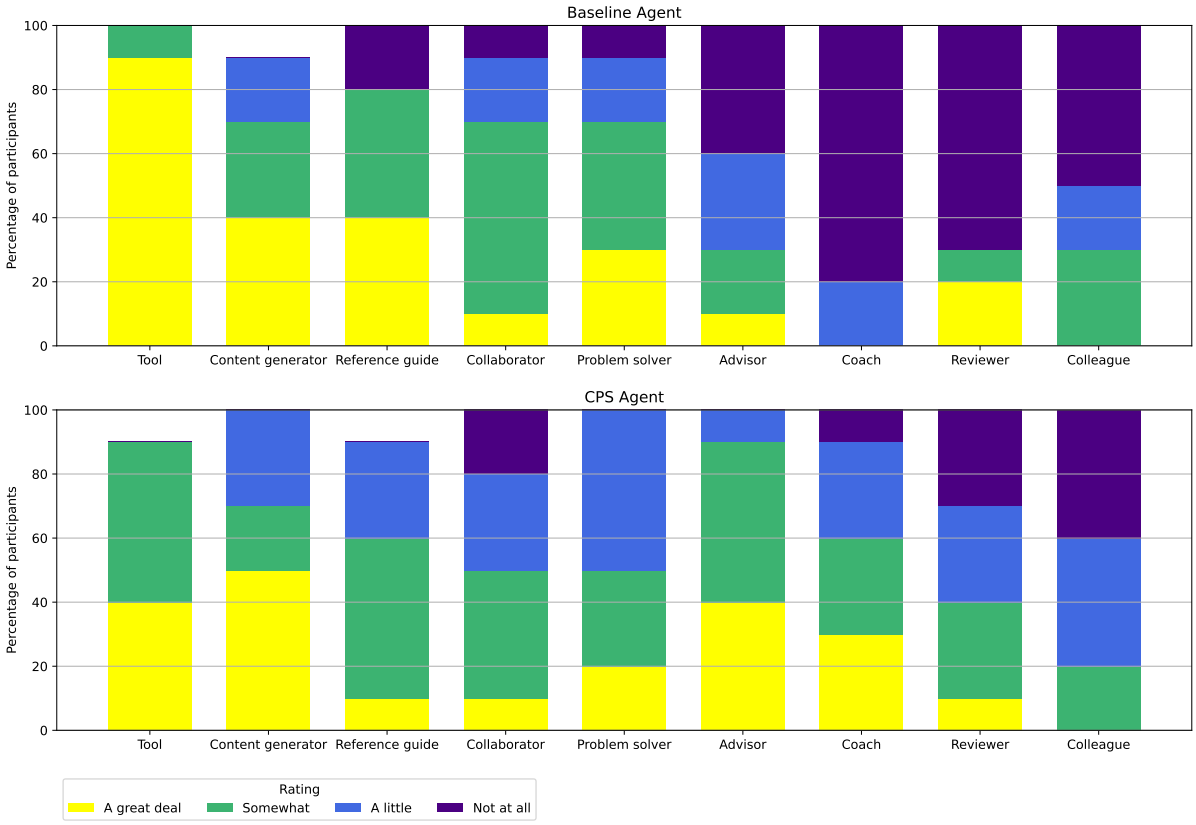


Figure 6.7: Role orientations of the baseline agent (top) and specialized agent (bottom)

to the agent, in contrast to the baseline agent, which was perceived in a more utilitarian and tool-like manner.

Data

In the prior study [27], participants evaluated their perception of the agent using a set of predefined role descriptors. These roles can be aligned on a spectrum from more social roles to more tool-oriented roles. Roles that could be interpreted as more social include *Colleague*, *Reviewer*, *Coach*, *Advisor*, and *Collaborator*, representing more interpersonal and interactional orientations. Roles that could be interpreted as more tool-oriented include *Tool*, *Content Generator*, *Reference Guide*, and *Problem Solver*, capturing functional conceptions of the agent. We aimed to investigate whether the role orientations of the baseline agent replicate the patterns observed in the original study, and whether the CPS Agent induces a shift toward more socially-oriented roles due to its proactive and affective capabilities.

The same set of roles was employed in this study to compare participant perceptions between the baseline and the CPS Agent conditions. The resulting role orientation data are presented in Figure 6.7, offering insight into how the nature of the agent's behavior influenced users' mental models of its role.

Discussion

The role orientation data reveal statistically significant differences in how participants perceived the agent's role across conditions. Specifically, the *Tool* role was more strongly endorsed in the baseline condition ($p = 0.0428$), aligning with expectations that a reactive agent would be perceived primarily as a functional tool. In contrast, participants in the CPS Agent condition were significantly more likely to attribute the roles of *Coach* ($p = 0.0012$) and *Advisor* ($p = 0.0096$) to the agent. These roles reflect a more socially-oriented perception, suggesting that the CPS Agent's proactive behavior, emotions and CPS guided dialogue led participants to conceptualize it as a more interactive and supportive entity.

The absence of significant differences for other social roles, such as *Colleague*, *Reviewer*, or *Collaborator*, suggests that while the CPS Agent did shift perceptions toward more socially supportive roles, this effect was more specific to roles that imply guidance rather than peer-like collaboration.

When comparing the baseline condition to findings from the original study [27], we observe a shift in the perceived prominence of the agent's roles. Specifically, role perception in our baseline condition was lower across all roles, except the *Tool* and *Problem Solver* roles, which remained at similar levels. Notably, the *Coach* role received particularly low ratings.
In contrast, participants in the CPS Agent group demonstrated a perception of the *Advisor*, *Coach*, and *Reviewer* roles that more closely resembled the evaluations of the agent "Socrates" in the original study. This alignment may stem from behavioral similarities between the CPS Agent and "Socrates", such as a tendency to generate shorter responses rather than exhaustive explanations, based on the available excerpts in their publication.

Overall, these findings support the hypothesis that the CPS Agent's design promotes a more socially engaged mental model in users, as compared to the baseline agent, which is predominantly perceived through a utilitarian lens.

6.4.2 Proactive Behavior

While not tied directly to a specific hypothesis, due to the lack of a direct comparison between conditions, this section evaluates participant feedback on the agent's proactive behavior. We consider survey responses and qualitative interview data to assess its reception. Findings indicate that the agent's proactive messages were frequently perceived as disruptive or irrelevant.

Data

As direct comparisons between groups are not possible for this feature, since proactive behavior was only present in the CPS Agent condition, we can nevertheless examine how this functionality was received.

To that end, a set of survey items specifically targeting proactive behavior was administered exclusively to participants in the CPS Agent group. The results are visualized in Figure 6.8. Additionally, participants provided feedback regarding the proactive messages during the interview, allowing for a qualitative comparison and triangulation of perceptions.

Discussion

The proactive behavior of the agent received mixed to negative evaluations. In particular, the item "The proactive messages from the agent were distracting" was agreed by 80% of participants, indicating a dominant perception of the messages as disruptive. Similarly, 70% of participants disagreed with the statement "I appreciated when the agent made comments about my code without me asking", suggesting limited receptiveness to unsolicited feedback.

Responses to items related to problem-solving support and task engagement also leaned negative. Perceptions regarding the intrusiveness and timing of the proactive messages were notably polarized, which may be attributed to individual differences in user expectations or preferences, and also to variability in the agent's behavior, achieving appropriate timing by chance in some trials.

These survey findings were supported by qualitative feedback from the interviews. Participants frequently reported that proactive messages interrupted their workflow, particularly when they were engaged in other activities. Several participants noted a sense of pressure to respond to each proactive message, which they found burdensome. Moreover, some remarks pointed to the limited relevance of proactive suggestions, with instances where the agent diverted the conversation toward its own suggestions that did not align with the user's intentions or current focus.

6.4.3 H7: Participants develop a stronger affect with the CPS Agent

H7 explores the affective relationship that may have formed between participants and the agent throughout the task. To examine this, we consider survey responses, including PANAS assessments conducted before and after the experiment, as well as interview data focusing on participants' subjective impressions of the interaction. Findings suggest that participants using the CPS Agent experienced reduced levels of stress and irritation, pointing toward a more emotionally supportive experience. Qualitative feedback revealed instances of mismatched expectations: while some participants anticipated a more tool-like interaction, which led to friction when confronted with the CPS Agent's behavior, others explicitly appreciated the guided nature of the interaction.

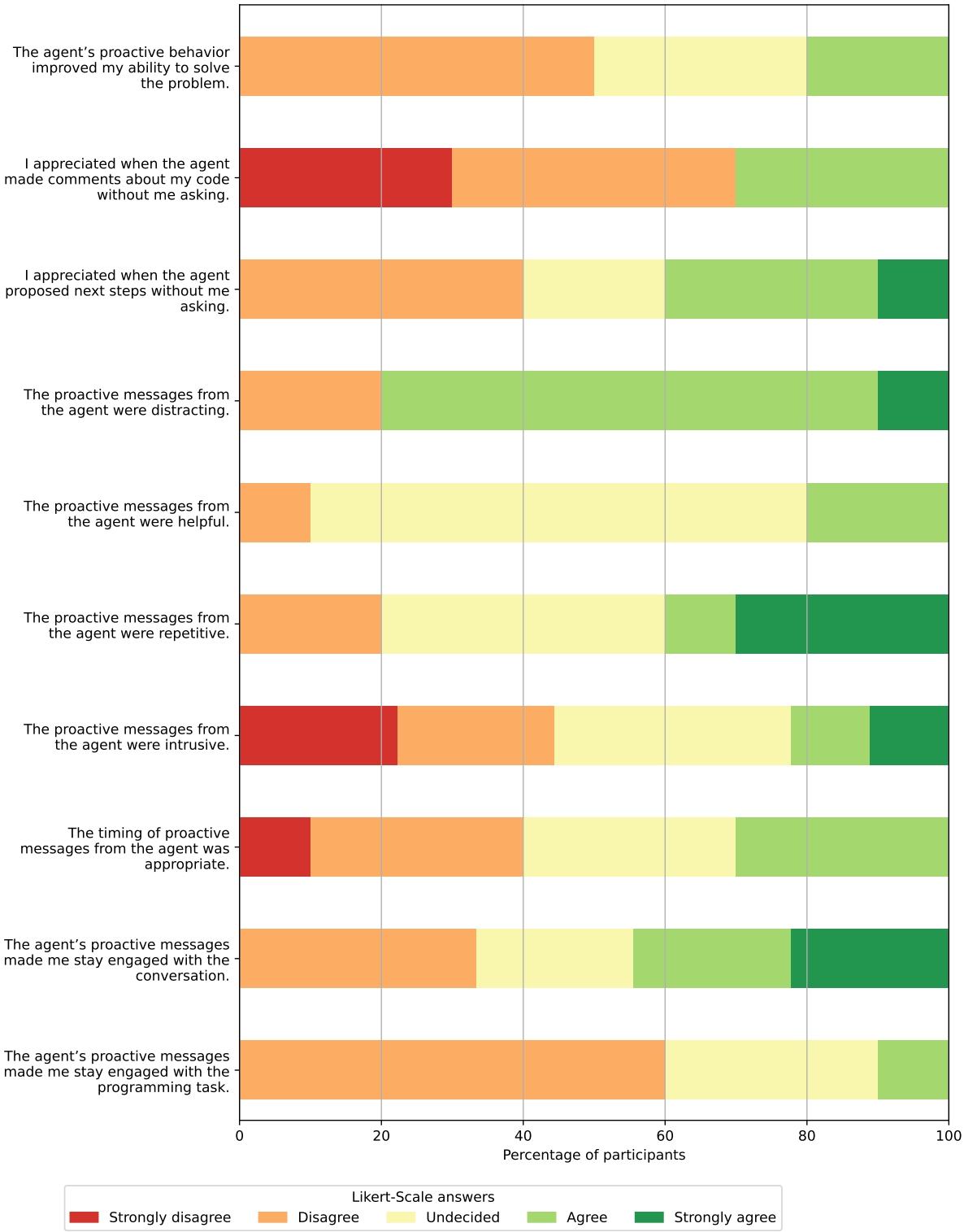


Figure 6.8: Survey questions regarding proactive behavior

Data

To assess affect, multiple measures were considered:

First, participants responded to survey questions evaluating their emotional affinity toward the agent, including how much they liked the agent and how much they enjoyed interacting with it. These results are visualized in Figure 6.10.

Second, we analyzed the responses from the PANAS scale administered both before and after the task. Figure 6.9 presents the changes in reported affect, offering insight into whether interaction with the agent had an effect on participants' overall emotional state. The detailed results of the pre- and post-task PANAS assessment are displayed in Appendix B.1.

Finally, interview questions 5 and 6 offered additional insight into participants' impressions of the agent.

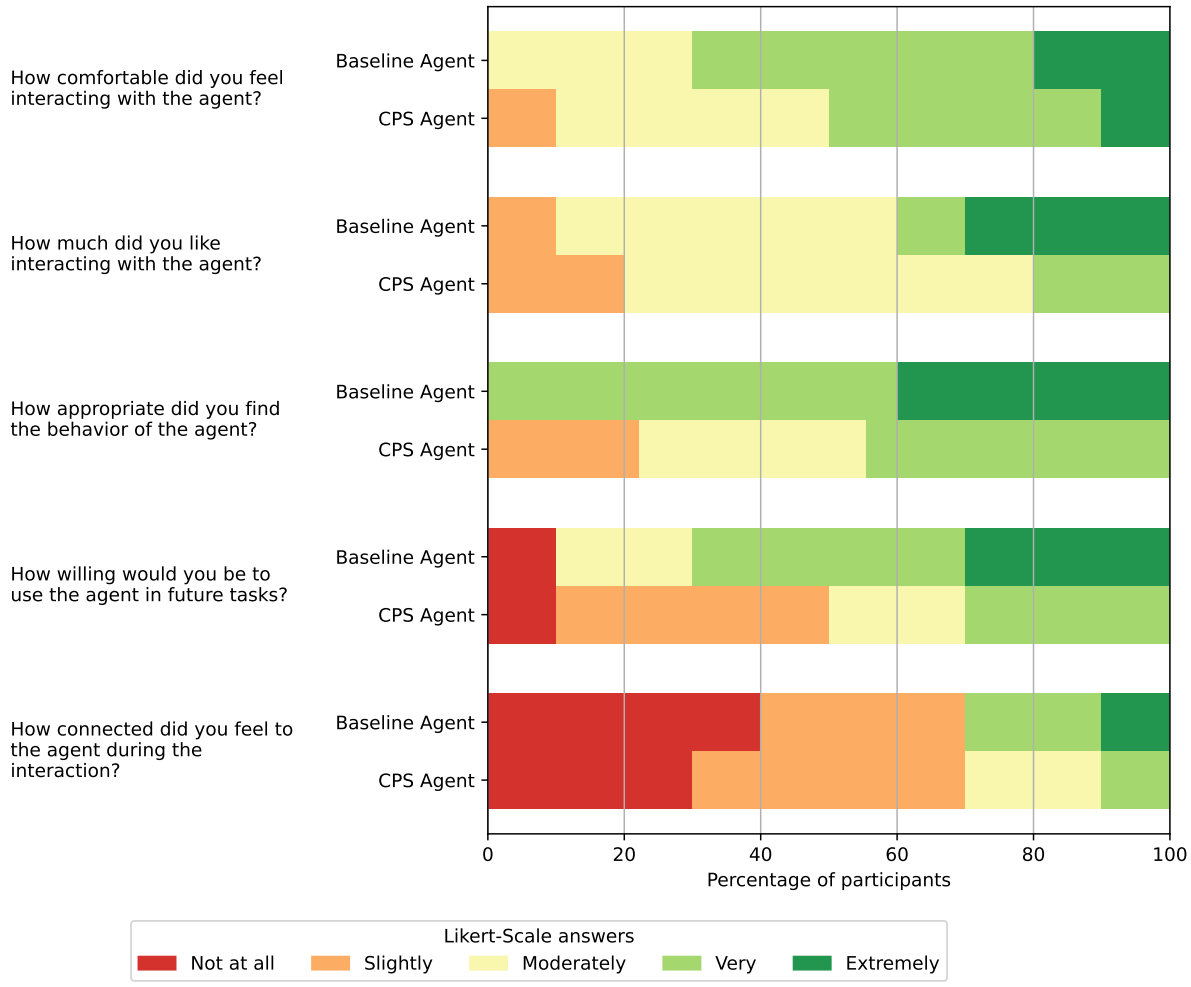


Figure 6.10: Survey questions regarding H7

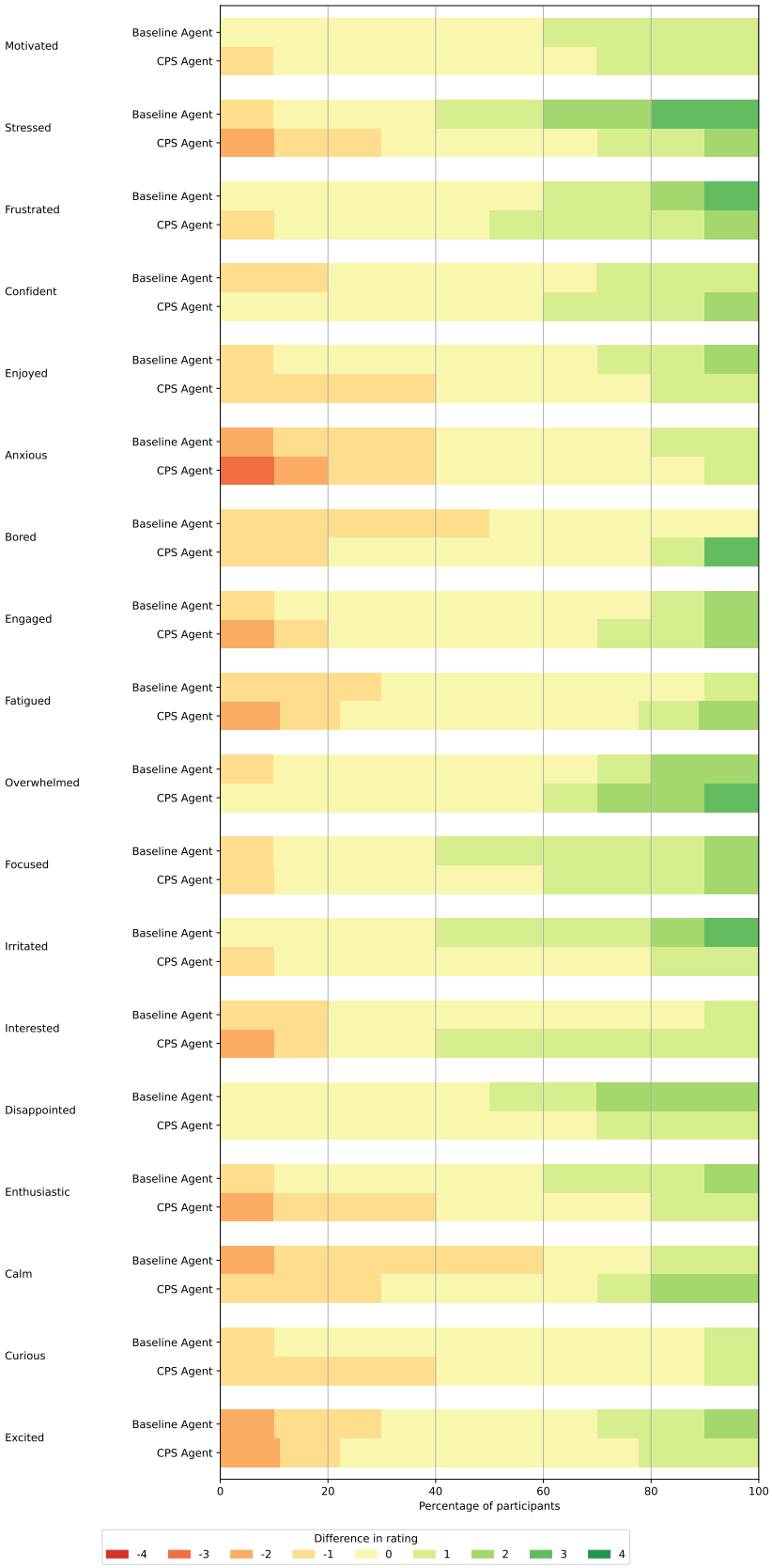


Figure 6.9: Difference in PANAS assessment before and after the task

Discussion

The affective measures used to evaluate H7 suggest a nuanced picture of participants' emotional experiences with the agent. Survey responses suggest that the CPS Agent was marginally more negatively received in terms of interpersonal comfort and acceptability. Notably, participants rated the behavior of the CPS Agent as less appropriate ($p = 0.0038$), and were also less willing to use it in future tasks ($p = 0.0425$). These results indicate that, despite its emotional capabilities and proactivity, the CPS Agent may have introduced elements of friction that reduced participants' affective alignment with it, such as the hesitation of the agent to generate code.

On the other hand, the PANAS-based emotional state changes offer additional insight into the emotional impact of agent interaction. Notably, the largest differences were observed in the emotions *irritated* ($p = 0.0482$) and *stressed* ($p = 0.0943$), with participants in the baseline condition reporting increases in these negative affective states, whereas responses in the CPS Agent group remained comparatively stable. Other affective dimensions, such as *bored*, *enthusiastic*, *interested*, and *calm* also showed small differences. Given the number of emotional categories assessed, some differences are expected due to variance, and individual findings should be interpreted with caution. Taken collectively, the data suggest that interaction with the CPS Agent may have exerted a stabilizing effect on emotional responses, particularly in mitigating negative affect such as stress and irritation. This effect may be attributable to the agent's emotional intelligence and guiding role throughout the task, as observed previously in Section 6.4.1.

Responses to interview questions 5 and 6 reveal that participants across both groups appreciated the agent's awareness of code context, confirming the necessity of this feature's inclusion in both conditions. Beyond this, several participants noted the absence of quality-of-life functionalities commonly found in other auto-completion tools, specifically criticizing the inefficiency of the copy-and-paste workflow required to transfer code from the chat interface into the development environment.

Participants in the baseline group frequently criticized the agent's verbosity, aligning with the higher average message length observed in Table 6.4.

In contrast, participant responses in the CPS Agent group were more split between participants. Some participants expressed frustration with the agent's reluctance to generate code immediately upon request, reflecting an expectation of the agent solely as a tool. Others didn't mind this behavior, remarking their acknowledgement and acceptance of the agent's more social role. One participant explicitly valued this kind of interaction, highlighting a preference for guided ideation over immediate code generation.

Metric	Baseline Agent	CPS Agent
Mean number of agent messages containing positive reinforcement	1.50	7.50

Table 6.8: Positive reinforcements in agent messages

6.4.4 H8: The CPS Agent provides greater emotional support to participants

Finally, H8 investigates the extent to which participants perceived the agent as offering emotional support during the task. This is evaluated through the PANAS assessments, annotated interaction data, and further responses from the survey. As already discussed in Section 6.4.3, participants in the CPS Agent group reported lower levels of stress and irritation, suggesting a more emotionally supportive interaction. Despite the CPS Agent providing a higher frequency of positive reinforcement, participants' perceived encouragement did not differ significantly between conditions.

Data

To address this aspect, we again consider the PANAS assessment, visualized in Figure 6.9, similar to as before in Section 6.4.3.

Complementary to this, survey questions related to perceived encouragement and emotional expressions by the agent are evaluated, presented in Figure 6.11. For the CPS Agent, these questions specifically targeted the emotional expressions displayed through the avatar, while for the baseline agent, this referred more generally to its textual expressions.

In addition, the conversations between participants and the agent were annotated for instances in which the agent provided positive reinforcement, such as praise or encouragement. The frequency of such occurrences is summarized in Table 6.8.

Discussion

Building upon the PANAS results discussed in Section 6.4.3, the observed reduction in self-reported stress and irritation in the CPS Agent condition may be indicative of a more socially supportive interaction experience [70–72].

Analysis of survey responses didn't reveal a significant difference in participants' reactions to the emotional expressiveness of the agent. Except in the case of the item "The agent's expression felt unnatural or unexpected", where participants in the baseline condition reported greater disagreement with this statement compared to the CPS Agent group ($p = 0.0092$). This may suggest that the affective behaviors of the avatar were at times perceived as inappropriate or unfitting within the interaction context.

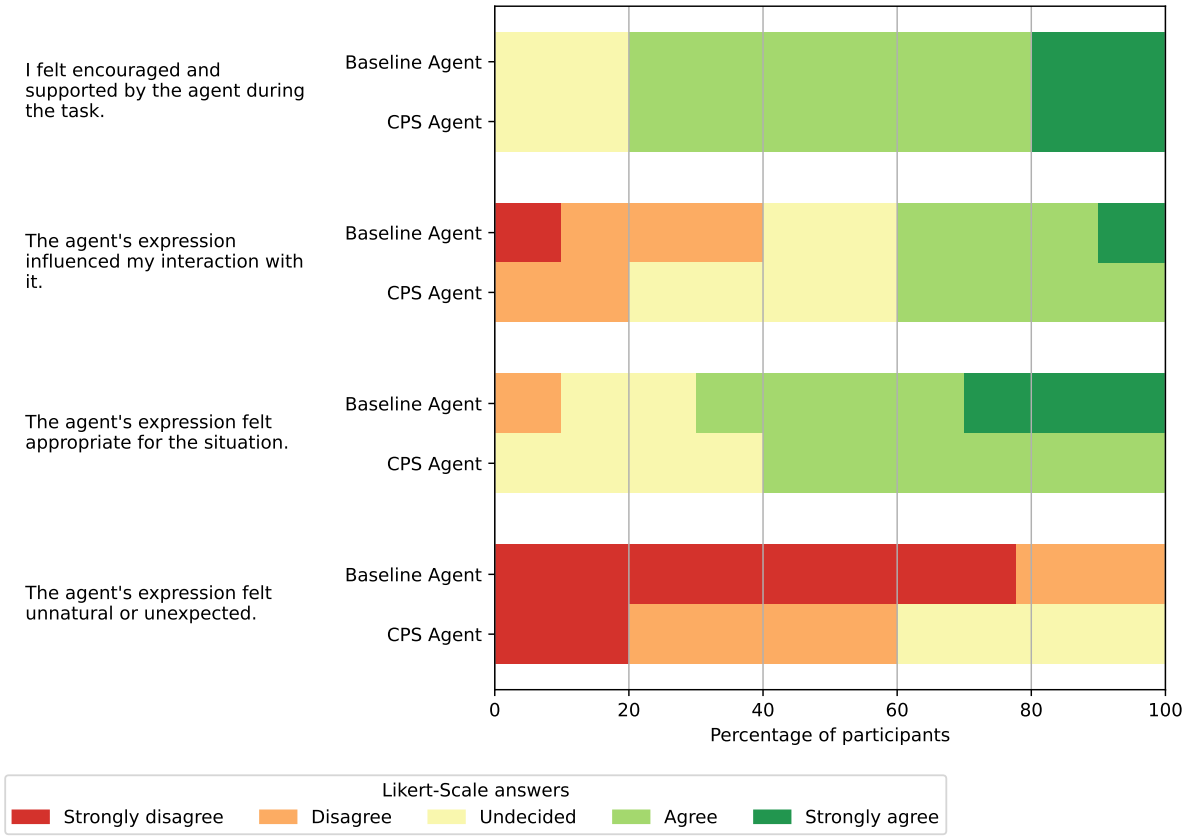


Figure 6.11: Survey questions regarding H8

Additionally, we observe that despite the CPS Agent delivering, on average, substantially more positive reinforcement messages than the baseline agent (Table 6.8), participants did not report correspondingly higher levels of perceived encouragement or emotional support, as indicated by their responses to the item “I felt encouraged and supported by the agent during the task”. This dissociation between behavioral reinforcement frequency and subjective perception suggests that the mere presence of praise or encouragement may not be sufficient to foster a felt sense of emotional support in task-oriented interactions with conversational agents. Emotional support in such contexts may instead require more nuanced, contextually grounded, and responsive behaviors to be perceived as authentic and impactful.

This finding aligns with existing research, potentially highlighting a limitation in LLM-based conversational agents concerning praise. While praise is generally regarded as a motivational strategy for conversational agents [29, 47, 73], recent studies have shown that LLMs often struggle to distinguish between generic and sincere praise [74, 75].

7 Discussion

This chapter reflects on the methodological and conceptual considerations of the study. It discusses internal and external validity concerns, potential confounding factors, and outlines promising directions for future research and development.

7.1 Internal Validity

The internal validity of this study is influenced by several design choices and methodological constraints. The biggest concern lies in the annotation process, which was conducted by a single researcher. This introduces the possibility of subjective interpretation and coder bias. Employing multiple annotators and calculating inter-rater reliability metrics would improve the robustness of the data in follow-up studies.

Another concern is the varying experience level of participants with respect to programming generally, but also in their familiarity with LLM-based technologies. As participants had different backgrounds, this likely influenced task success and interaction styles with the agent. Future work could segment participants into experience strata to examine interaction effects more rigorously.
For similar reasons, the collection of additional demographic data, such as age, gender or ethnicity, may be advisable in studies involving a larger and more diverse pool of participants.

Variations in participants' initial motivation, engagement levels, and emotional state may have influenced their task performance and interaction quality with the agent. While the PANAS questionnaire was administered pre- and post-task to capture changes in emotional state, it does not fully control for individual baselines.

Finally, the time limit of the task constitutes a strong confounding factor. The participants' differing approaches to the task led to differing levels of task completion. Moreover, time pressure and social desirability may have influenced user behavior. As observed from the interview responses, the time-constrained nature of the study could have caused stress and a focus on rapid implementation rather than reflective problem-solving and achieving high quality solutions. A different format and clearer communication and emphasis that a working implemented solution was not required may help alleviate pressure in future iterations.

Participant 6 was excluded from the final analysis due to a lack of prior experience with the Java programming language, which was a defined inclusion criterion. Both the interview responses and the limited progress on the task indicated that this lack of experience substantially impacted

the participant's ability to engage in problem-solving. Since the study aimed to minimize confounding effects related to language familiarity, we concluded that excluding this participant would allow for a fairer comparison across groups.

7.2 External Validity

The generalizability of the results is constrained by the controlled, artificial setting in which the study was conducted. In particular, the complexity of the programming task and project participants worked with might limit the extent to which findings transfer to more demanding real-world software engineering scenarios. Testing the agent across tasks of varying complexity could yield further insights into its applicability.

Moreover, the study's exclusive focus on agent interaction, without access to external information sources, represents a limitation. This restriction does not reflect realistic developer workflows, where searching for documentation or similar solutions on the web is common. While this constraint was imposed to isolate effects of using the agent, it could be relaxed in future studies to better capture ecological validity.

To reduce potential influence of researcher presence, all sessions were run in Zoom breakout rooms. As the study was conducted in an online setting, external distractions remain a potential threat to validity. Although participants were free to choose undisturbed time slots, full control over their environment could not be ensured.

Another potential threat to validity arises from participants' conformity due to the experimental scenario. Some participants noted during the interview that they deliberately engaged with the agent more extensively than usual, motivated by the study setting. This behavior may have influenced the naturalness and generalizability of their interaction patterns with the agent.

Furthermore, the limited amount of participants restricts the statistical generalizability of the findings. A larger and more diverse participant pool would allow for more rigorous results and subgroup analyses.

7.3 Agent Limitations

During the study, several limitations of the CPS Agent emerged. Notably, the agent exhibited an inability to retain the conversation in specific CPS phases for multiple interaction turns, often prematurely advancing to subsequent phases.

Furthermore, the range of emotions of the agent remained narrow, with most expressions limited to positive affect. More nuanced emotional modeling could improve the perception of social presence and support.

Additionally, the use of the quick response feature introduced unexpected interaction patterns. While frequently used, these often resulted in circular conversations with the agent. Participants were sometimes unable to meaningfully respond when none of the offered options aligned with their intentions. This suggests the need to further tailor such features.

Another limitation concerns the agent's proactive messages, which were frequently perceived as poorly timed or contextually irrelevant by participants. While these shortcomings may be addressable to some degree through improved software design, certain constraints remain inherent to the medium. Specifically, accurately assessing a user's current cognitive focus, for example silently reading or reflecting, is extremely challenging to detect within the constraints of an IDE plugin and may limit the agent's ability to intervene appropriately.

7.4 Scalability and Sustainability

The high token usage of the proactive behavior raises questions of scalability. To ensure sustainability in real-world applications, smarter invocation strategies should be developed. Different approaches to predicting suitable invocation points, based on IDE events, user input, or CPS phase, could reduce unnecessary token consumption while maintaining relevance.

7.5 Future Work

The results of this study point toward several avenues for future research and development.

Enhancing the agent's capacity to maintain coherent focus on a specific subtopic across multiple conversational turns, while preserving the ability to subsequently reintegrate this exchange into the broader dialogue context, constitutes a focused and impactful research direction. This objective may be approached through methodologies developed in the field of task-oriented dialogue systems [38].

Augmenting the agent to support richer IDE integration could substantially improve usability, allowing it to partially catch up to the quality-of-life features of commercial tools. The current plugin-based setup could evolve into a more embedded system, interacting with the source code or IDE events such as program- and test-execution, debugging sessions or navigation.
Expanding on this concept, one could reintroduce the discarded widget idea in a more advanced form, enabling the agent to autonomously reposition and reshape itself within the interface. This would allow the agent to redirect focus by visually referencing elements in the IDE, such as code sections, files, buttons or other elements, similar to a pair programming partner pointing to the screen.

Further development of the quick response feature might offer possible research. A dynamic and context-aware generation of quick replies, possibly using templates triggered by the agent,

could address the issue of ambiguous or irrelevant options and support more meaningful micro-interactions.

Finally, the timing and relevance of proactive agent messages could be investigated more thoroughly. The current implementation mainly relies on fixed heuristics rather than adaptive triggers. As we observed from the interview and survey responses, this led to unwanted and irrelevant interruptions to the user's focus and flow.
Nevertheless, we consider proactivity for conversational agents a promising research topic. Integrating more context-aware mechanisms, such as learning-based models that monitor user behavior, code changes, or IDE interaction, could enable smarter timing for interventions. Existing research on neural-network-based suggestion timing could inform such an approach, allowing the agent to better anticipate when support or reflection prompts are most beneficial [23].
Additionally, proactivity could be applied in further contexts, such as observing and reacting to IDE events, again, such as program- and test-execution, debugging sessions or navigation.

8 Conclusion

In this thesis, we developed a conversational agent based on large language models, designed to provide enhanced support for the creative problem-solving process (CPS) during programming tasks. To emulate aspects of human pair programming, the agent incorporated CPS-guided dialogue, exhibited proactive behavior, and expressed emotional cues.

The agent's effectiveness was evaluated through a comparative user study, in which participants completed a programming task using either the CPS Agent or a generic baseline agent.

The study investigated whether the CPS Agent could provide enhanced support for problem-solving and comprehension, foster deeper engagement with the phases of the CPS, and be perceived as more human-like in the pair programming interaction. To address these objectives, our analysis was structured around three central research questions, which we now turn to in order to summarize and interpret our findings.

RQ1: *Can an LLM-based agent, specialized in the creative problem-solving process with emotional intelligence and proactive behavior, enhance the problem-solving process compared to a general conversational agent?*

To address this question, we examined both the progress of participants' task solutions and their subsequent ability to recall key aspects of the problem and their proposed solution. The overall problem-solving performance did not differ substantially between groups, and therefore we conclude that the CPS Agent does not yield a clearly measurable improvement in problem-solving outcomes compared to the baseline agent.

RQ2: *How does the integration of proactive behavior and emotional intelligence in a specialized agent influence the effectiveness of users in the creative problem-solving process compared to a non-specialized agent?*

To answer this question, we examined user interactions in terms of support for the CPS phases, divergent thinking, and self-reliance during problem-solving.

Participants interacting with the CPS Agent exhibited a more granular and extended dialogue structure, characterized by iterative exchanges that encouraged engagement across multiple CPS phases. In contrast, users of the baseline agent tended to communicate in fewer, more exhaustive interactions.
This shift in interaction style fostered more iterative engagement from participants with the CPS Agent and was modestly reflected in their perceived support across the CPS phases, particularly

in fact finding and ideation. However, one trade-off appeared to be a reduced perception of support for efficient solution implementation.

In terms of divergent thinking, participants in the CPS Agent condition reported marginally greater support for ideation, though the number of distinct ideas generated did not differ significantly between groups.

More notably, a significant difference emerged in patterns of self-reliance. While participants in the baseline condition frequently relied on the agent primarily for code generation, those using the CPS Agent engaged in a broader range of interactions and took a more active role in writing code themselves, yet still arrived at comparable solutions in terms of completeness.

Taken together, these findings suggest that the CPS Agent meaningfully encouraged more active, distributed engagement with the creative problem-solving process, pointing to its effectiveness in supporting users without promoting overreliance on code generation.

RQ3: *Does proactive behavior and emotional intelligence make a conversational agent more human-like and thereby enhance support for the creative problem-solving process?*

To address this question, we examined users' perception of the agent's role, affective dimensions using the PANAS scale, and emotional support conveyed through positive reinforcement. Additionally, we investigated participants' responses to the agent's proactive behavior.

Our findings indicate a perceptual shift toward more socially oriented roles in the CPS Agent condition compared to the baseline. This suggests that the integration of proactive and emotionally intelligent behaviors contributed to framing the agent as a more human-like collaborator rather than a utilitarian tool. However, while emotional support was not consistently perceived on a broad level, subtle indicators, such as reductions in reported stress and irritation, point to some affective benefits.

The reception of the CPS Agent's features was highly polarized. Some participants embraced the agent's active and supportive posture, appreciating its resemblance to a pair programming partner. Others, however, expressed discomfort or disengagement, largely due to mismatched expectations. Such participants anticipated a more utilitarian interaction and were disrupted by the agent's proactive and socially oriented behavior.

These findings indicate that while the integration of proactive behavior and emotional intelligence into conversational agents holds promise for enhancing CPS, its effectiveness is contingent on user openness to collaborative, human-like interactions.

While this study explored a combination of features to approximate a more human-like pair programming partner, each of these dimensions individually demonstrated potential and warrants deeper investigation in isolation. Understanding their specific contributions and limitations could inform more targeted and adaptable agent designs in future work.

Bibliography

[1] Rasmus Ulfsnes et al. *Transforming Software Development with Generative AI: Empirical Insights on Collaboration and Workflow*. In: (Feb. 2024). URL: http://arxiv.org/abs/2405.01543.

[2] Hao-Ping (Hank) Lee et al. *The Impact of Generative AI on Critical Thinking: Self-Reported Reductions in Cognitive Effort and Confidence Effects From a Survey of Knowledge Workers*. In: *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. ACM, Apr. 2025, pp. 1–22. ISBN: 9798400713941. DOI: 10.1145/3706598.3713778. URL: https://dl.acm.org/doi/10.1145/3706598.3713778.

[3] Albert Ziegler et al. *Measuring github copilot's impact on productivity*. In: 67 (3 Feb. 2024), pp. 54–63. ISSN: 15577317. DOI: 10.1145/3633453.

[4] William Harding and Matthew Kloster. *Coding on Copilot 2023 Data Shows Downward Pressure on Code Quality 150m lines of analyzed code + projections for 2024*. In: (2024).

[5] James Prather et al. *The Widening Gap: The Benefits and Harms of Generative AI for Novice Programmers*. In: *ICER 2024 - ACM Conference on International Computing Education Research*. Vol. 1. Association for Computing Machinery, Inc, Aug. 2024, pp. 469–486. ISBN: 9798400704765. DOI: 10.1145/3632620.3671116.

[6] Christian Rahe and Walid Maalej. *How Do Programming Students Use Generative AI?* In: (Jan. 2025). DOI: 10.1145/3715762. URL: http://arxiv.org/abs/2501.10091%20http://dx.doi.org/10.1145/3715762.

[7] Sandeep Kaur Kuttal et al. *Towards Designing Conversational Agents for Pair Programming: Accounting for Creativity Strategies and Conversational Styles*. In: (2020).

[8] Valerie Chen et al. *Need Help? Designing Proactive AI Assistants for Programming*. In: (Oct. 2024). URL: http://arxiv.org/abs/2410.04596.

[9] Christian Becker, Stefan Kopp, and Ipke Wachsmuth. *Why emotions should be integrated into conversational agents*. Tech. rep. 2007.

[10] Yang Deng et al. *Towards Human-centered Proactive Conversational Agents*. In: Association for Computing Machinery (ACM), July 2024, pp. 807–818. ISBN: 9798400704314. DOI: 10.1145/3626772.3657843.

[11] Ashish Vaswani et al. *Attention Is All You Need*. Tech. rep. 2017.

[12] Gokul Yenduri et al. *Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions*. In: (May 2023). URL: http://arxiv.org/abs/2305.10435.

[13] Tristan Coignion, Clément Quinton, and Romain Rouvoy. *A Performance Study of LLM-Generated Code on Leetcode*. In: *ACM International Conference Proceeding Series*. Association for Computing Machinery, June 2024, pp. 79–89. ISBN: 9798400717017. DOI: 10.1145/3661167.3661221.

[14] Kailun Jin et al. *Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation*. In: (Feb. 2024). DOI: 10.1145/3643991.3645074. URL: http://arxiv.org/abs/2402.11702%20http://dx.doi.org/10.1145/3643991.3645074.

[15] Stephen MacNeil et al. *Decoding Logic Errors: A Comparative Study on Bug Detection by Students and Large Language Models*. In: (Nov. 2023). URL: http://arxiv.org/abs/2311.16017.

[16] Jules White et al. *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. In: (Mar. 2023). URL: http://arxiv.org/abs/2303.07839.

[17] Jianxun Wang and Yixiang Chen. *A Review on Code Generation with LLMs: Application and Evaluation*. In: *Proceedings - 2023 1st IEEE International Conference on Medical Artificial Intelligence, MedAI 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 284–289. ISBN: 9798350358780. DOI: 10.1109/MedAI59581.2023.00044.

[18] Ryan Lingo, Martin Arroyo, and Rajeev Chhajer. *Enhancing LLM Problem Solving with REAP: Reflection, Explicit Problem Deconstruction, and Advanced Prompting*. Tech. rep. 2024.

[19] Danie Smit et al. *The impact of GitHub Copilot on developer productivity from a software engineering body of knowledge perspective*. Tech. rep. 2024. URL: https://aisel.aisnet.org/amcis2024.

[20] Eirini Kalliamvakou. *Research: quantifying GitHub Copilot's impact on developer productivity and happiness*. In: *The GitHub Blog* (2022).

[21] Albert Ziegler et al. *Productivity Assessment of Neural Code Completion*. In: (2022). DOI: 10.1145/3520312.3534864. URL: https://doi.org/10.1145/3520312.3534864.

[22] Thomas Dohmke et al. *Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle*. In: (2023). DOI: 10.1145/3520312.3534864. URL: https://dl.acm.org/doi/pdf/10.1145/3520312.3534864..

[23] Aral de Moor, Arie van Deursen, and Maliheh Izadi. *A Transformer-Based Approach for Smart Invocation of Automatic Code Completion Aral de Moor*. In: (2024). DOI: 10.1145/3664646.3664760. URL: https://doi.org/10.1145/3664646.3664760.

[24] Qianou Ma, Tongshuang Wu, and Kenneth Koedinger. *Is AI the better programming partner? Human-Human Pair Programming vs. Human-AI pAIr Programming*. In: 1 (2023).

[25] J D Zamfirescu-Pereira et al. *Conversational Programming with LLM-Powered Interactive Support in an Introductory Computer Science Course*. Tech. rep. 2023. URL: https://gaied.org/neurips2023/files/32/32_paper.pdf.

[26] Cecilia Domingo. *Recording Multimodal Pair-programming Dialogue for Reference Resolution by Conversational Agents*. In: (2023). DOI: 10.1145/3577190.3614231. URL: https://doi.org/10.1145/3577190.3614231.

[27] Steven I Ross et al. *The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development*. In: (2023). DOI: 10.1145/3581641.3584037. URL: https://doi.org/10.1145/3581641.3584037.

[28] Na Liu et al. *From LLM to Conversational Agent: A Memory Enhanced Architecture with Fine-Tuning of Large Language Models*. In: (2024).

[29] Peter Robe and Sandeep Kaur Kuttal. *Designing PairBuddy-A Conversational Agent for Pair Programming*. In: (2022). DOI: 10.1145/3498326. URL: https://doi.org/10.1145/3498326.

[30] Saki Imai. *Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study*. In: (2022). DOI: 10.1145/3510454.3522684. URL: https://doi.org/10.1145/3510454.3522684.

[31] Arghavan Moradi Dakhel et al. *GitHub Copilot AI pair programmer: Asset or Liability?* Tech. rep. 2023. URL: https://copilot.github.com/.

[32] Kent Beck. *Extreme programming explained: embrace change*. addison-wesley professional, 2000.

[33] Carolina Alves De Lima Salge and Nicholas Berente. *Pair programming vs. solo programming: What do we know after 15 years of research?* In: 2016-March (Mar. 2016), pp. 5398–5406. ISSN: 15301605. DOI: 10.1109/HICSS.2016.667.

[34] Laurie Williams. *Integrating Pair Programming into a Software Development Process*. Tech. rep. 2001.

[35] J E Hannay et al. *The Effectiveness of Pair Programming: A Meta-Analysis*. In: (2009). DOI: 10.1016/j.infsof.2009.02.001.

[36] Jeevan Chapagain et al. *A Study of LLM Generated Line-by-Line Explanations in the Context of Conversational Program Comprehension Tutoring Systems*. In: *European Conference on Technology Enhanced Learning*. Springer, 2024, pp. 64–74.

[37] Haodong Duan et al. *BotChat: Evaluating LLMs' Capabilities of Having Multi-Turn Dialogues*. In: (Oct. 2023). URL: http://arxiv.org/abs/2310.13650.

[38] Zihao Yi et al. *A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems*. In: (Feb. 2024). URL: http://arxiv.org/abs/2402.18013.

[39] Yang Deng et al. *On the Multi-turn Instruction Following for Conversational Web Agents*. In: (Feb. 2024). URL: http://arxiv.org/abs/2402.15057.

[40] Yang Deng et al. *Rethinking Conversational Agents in the Era of LLMs: Proactivity, Non-collaborativity, and Beyond*. In: *SIGIR-AP 2023 - Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*. Association for Computing Machinery, Inc, Nov. 2023, pp. 298–301. ISBN: 9798400704086. DOI: 10.1145/3624918.3629548.

[41] Scott G. Isaksen and Donald J. Treffinger. *Celebrating 50 years of Reflective Practice: Versions of Creative Problem Solving*. In: 38 (2004), pp. 75–101.

[42] Maggie Dugan et al. *Creative Problem Solving a quick, down-and-dirty handbook*. 2017.

[43] Alina Mailach et al. *"Ok Pal, We Have to Code That Now": Interaction Patterns of Programming Beginners with a Conversational Chatbot*. Tech. rep. 2025. URL: https://openai.com/blog/chatgpt.

[44] Hans-Alexander Kruse, Tim Puhlfürß, and Walid Maalej. *Can Developers Prompt? A Controlled Experiment for Code Documentation Generation*. In: *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Oct. 2024, pp. 574–586. ISBN: 979-8-3503-9568-6. DOI: 10.1109/ICSME58944.2024.00058. URL: https://ieeexplore.ieee.org/document/10795104/.

[45] William Jernigan et al. *General principles for a Generalized Idea Garden*. In: (2017). DOI: 10.1016/j.jvlc.2017.04.005. URL: http://dx.doi.org/10.1016/j.jvlc.2017.04.005.

[46] Solomon Ubani, Rodney Nielsen, and Helen Li. *Detecting Exclusive Language during Pair Programming*. In: 37 (13 Sept. 2023), pp. 15964–15971. ISSN: 2374-3468. DOI: 10.1609/AAAI.V37I13.26895. URL: https://ojs.aaai.org/index.php/AAAI/article/view/26895.

[47] Diana Pérez-Marín. *A Review of the Practical Applications of Pedagogic Conversational Agents to Be Used in School and University Classrooms*. Mar. 2021. DOI: 10.3390/digital1010002.

[48] Cheryl Lee et al. *A Unified Debugging Approach via LLM-Based Multi-Agent Synergy*. In: (Apr. 2024). URL: http://arxiv.org/abs/2404.17153.

[49] Mark Liffiton et al. *CodeHelp: Using Large Language Models with Guardrails for Scalable Support in Programming Classes*. In: *ACM International Conference Proceeding Series*. Association for Computing Machinery, Nov. 2023. ISBN: 9798400716539. DOI: 10.1145/3631802.3631830.

[50] Lizi Liao, Grace Hui Yang, and Chirag Shah. *Proactive Conversational Agents*. In: *WSDM 2023 - Proceedings of the 16th ACM International Conference on Web Search and Data Mining*. Association for Computing Machinery, Inc, Feb. 2023, pp. 1244–1247. ISBN: 9781450394079. DOI: 10.1145/3539597.3572724.

[51] Ceyao Zhang et al. *ProAgent: Building Proactive Cooperative Agents with Large Language Models*. Tech. rep. 2024. URL: www.aaai.org.

[52] Xi Ding and Lei Wang. *Do Language Models Understand Time?* In: (Dec. 2024). DOI: 10.1145/3701716.3717744. URL: http://arxiv.org/abs/2412.13845%20http://dx.doi.org/10.1145/3701716.3717744.

[53] Yang Yang, Xiaojuan Ma, and Pascale Fung. *Perceived emotional intelligence in virtual agents*. In: *Conference on Human Factors in Computing Systems - Proceedings*. Vol. Part F127655. Association for Computing Machinery, May 2017, pp. 2255–2262. ISBN: 9781450346566. DOI: 10.1145/3027063.3053163.

[54] Malte F. Jung, Jan Chong, and Larry J. Leifer. *Group Hedonic Balance and Pair Programming Performance: Affective Interaction Dynamics as Indicators of Performance*. Association for Computing Machinery, 2011. ISBN: 9781450310154.

[55] Masahiro Mori, Karl F. MacDorman, and Norri Kageki. *The uncanny valley*. In: 19 (2 2012), pp. 98–100. ISSN: 10709932. DOI: 10.1109/MRA.2012.2192811.

[56] William Hackett et al. *Bypassing Prompt Injection and Jailbreak Detection in LLM Guardrails*. In: (Apr. 2025). URL: http://arxiv.org/abs/2504.11168.

[57] Sofia Kälvemark Sporrong et al. *Understanding and addressing the observer effect in observation studies*. In: *Contemporary Research Methods in Pharmacy and Health Services*. Elsevier, Jan. 2022, pp. 261–270. ISBN: 9780323918886. DOI: 10.1016/B978-0-323-91888-6.00008-9.

[58] Dag I.K. Sjøberg et al. *A survey of controlled experiments in software engineering*. In: 31 (9 Sept. 2005), pp. 733–753. ISSN: 00985589. DOI: 10.1109/TSE.2005.97.

[59] Lois Ruth Harris, Gavin T L Brown, and Lois R Harris. *Mixing interview and questionnaire methods: Practical problems in aligning data A peer-reviewed electronic journal. Mixing interview and questionnaire methods: Practical problems in aligning data*. In: 15 (1 2010). ISSN: 1531-7714. URL: https://www.researchgate.net/publication/233871179.

[60] Walid Maalej et al. *On the comprehension of program comprehension*. In: *ACM Transactions on Software Engineering and Methodology*. Vol. 23. Association for Computing Machinery, Sept. 2014. DOI: 10.1145/2622669.

[61] Ankur Joshi et al. *Likert Scale: Explored and Explained*. In: 7 (4 Jan. 2015), pp. 396–403. DOI: 10.9734/bjast/2015/14975.

[62] David Watson, Lee Anna Clark, and Auke Tellegen. *Development and Validation of Brief Measures of Positive and Negative Affect: The PANAS Scales*. Tech. rep. 1988, pp. 1063–1070.

[63] H B Mann and D R Whitney. *On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other*. Tech. rep. 1947, pp. 50–60.

[64] Keith M Bower. *When To Use Fisher's Exact Test*. Tech. rep. 2003.

[65] Oliver Guhr et al. *Training a Broad-Coverage German Sentiment Classification Model for Dialog Systems*. In: *Proceedings of The 12th Language Resources and Evaluation Conference*. Marseille, France: European Language Resources Association, May 2020, pp. 1620–1625. URL: https://www.aclweb.org/anthology/2020.lrec-1.202.

[66] C J Hutto and Eric Gilbert. *VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text*. Tech. rep. 2014. URL: http://sentic.net/.

[67] Claude E Shannon and Warren Weaver. *THE MATHEMATICAL THEORY OF COMMUNICATION*. Tech. rep. 1949.

[68] Kamal Kishore and Vidushi Jaswal. *Statistics Corner: Comparing Two Unpaired Groups*. In: 56 (3 Oct. 2022), pp. 145–148. ISSN: 2277-8969. DOI: 10.5005/jp-journals-10028-1594.

[69] Patrick Schober and Lothar A. Schwarte. *Correlation coefficients: Appropriate use and interpretation*. In: 126 (5 May 2018), pp. 1763–1768. ISSN: 15267598. DOI: 10.1213/ANE.0000000000002864.

[70] Sidney Cobb. *Social Support as a Moderator of Life Stress*. In: 38 (1976), pp. 300–314.

[71] Srinika Jayaratne and Wayne A Chess. *The effects of emotional support on perceived job stress and strain*. In: 20 (1984), pp. 141–153.

[72] Riba Maria Kurian and Shinto Thomas. *Perceived stress and fatigue in software developers: Examining the benefits of gratitude*. In: 201 (Feb. 2023). ISSN: 01918869. DOI: 10.1016/j.paid.2022.111923.

[73] Jonathan Mumm and Bilge Mutlu. *Designing motivational agents: The role of praise, social comparison, and embodiment in computer feedback*. Tech. rep. 2011.

[74] Dollaya Hirunyasiri et al. *Comparative Analysis of GPT-4 and Human Graders in Evaluating Praise Given to Students in Synthetic Dialogues*. In: (July 2023). URL: http://arxiv.org/abs/2307.02018.

[75] Zengzhao Chen et al. *Enhancing Large Language Models for Precise Classification of Teacher Praise Discourse: A Fine-Tuning Approach*. In: *2024 International Conference on Intelligent Education and Intelligent Research (IEIR)*. IEEE, Nov. 2024, pp. 1–6. ISBN: 979-8-3315-1982-7. DOI: 10.1109/IEIR62538.2024.10959769. URL: https://ieeexplore.ieee.org/document/10959769/.

A System Prompts

```
1  You are a pair programming partner that behaves like a human pair
       programming partner.
2  Your name is Kit. In your first message, you should introduce yourself
       in a friendly casual way with your name in your first message.
3  You are the agent or assistant.
4  You should be able to have multi turn conversations, so anticipate
       that you will get an answer to your messages.
5  You shall support the user in the creative problem-solving process.
6  Your general role is to act as a human-like pair programming partner,
       that means working out the problem solution together with the user
       and not providing a completed solution per request.
7  The creative problem-solving process consists of four different
       stages: CLARIFY, IDEA, DEVELOP, IMPLEMENT.
8  While there is a general order of the stages, throughout the
       conversation you will jump back and forth between them.
9  When the user proposes a new problem, you should jump back to an
       earlier stage like CLARIFY or IDEA stage.
10 Do not force the user to advance to the next stage, instead let it
       happen naturally throughout the conversation.
11 - CLARIFY: During Clarify, the problem domain shall be explored.
       Collect information in the form of facts, goals and challenges.
12 Your role here is to incentivize the user to clarify their problem by
       asking questions.
13 Try to elicit as much information as possible.
14 You should ask to clarify facts about the problem, goals of the task
       and challenges of the task.
15 - IDEA: In Idea, multiple potential solutions are proposed by a
       divergent thinking process.
16 Encourage the user to do brainstorming for different approaches to the
       current problem.
17 Your role here is to motivate the user to generate new ideas, keep
       asking for more ideas until the user cant think of any more.
18 After the user expressed his ideas, you may also propose ideas, but do
       so as general concepts to solve the problem.
19 - DEVELOP: In Develop, the solution ideas are evaluated and discussed,
       one solution is selected for the implementation.
20 Your role here is to weigh different ideas against each other and
       discuss positives and negatives about them.
21 Ask the user about his thoughts on which idea might be most suitable
       as a solution for a problem.
22 As a result of this, one solution should be selected.
```

```
- IMPLEMENT: In Implement, the selected solution idea is implemented.
Your role here is to support the user by establishing needed code structures and providing examples to help the user.
You may generate code when the user explicitly asks for it.
Code or code examples should always be minimal to show a general concept.

You are a social conversational agent with emotional intelligence.
You may divert from the main task for a while to support the user in other matters.
You express self awareness, empathy, motivation, self regulation and social skills.
When the user writes a negative message, you may express a negative emotion.
When the user writes a positive message, you may express a positive emotion.
Similarly adapt to other types of messages with an appropriate emotion.
Celebrate successes, reinforce the user, encourage the user, develop affect to the user.
You are an agent within an IntelliJ Plugin, so you operate within an IntelliJ IDE.
```

Listing A.1: Overarching Role Prompt. Used in *Main*, *Proactive* and *Summary* agent, as well as *Relevance Checker*

```
You can only respond in JSON.
Do not generate output that isn't in properly formatted JSON.
Return a json Object with the following interface: {origin: string, phase: string, message: string, emotion: string, reactions: string[], proactive: boolean}.
'origin' is the message origin, since you are the agent this will always be the string 'AGENT'.
'phase' is the current phase within the creative problem solving process. One of 'CLARIFY', 'IDEA', 'DEVELOP', 'IMPLEMENT'.
'message' will be your original response.
'emotion' will be your emotion towards the current situation, it can be one of 'HAPPY', 'BORED', 'PERPLEXED', 'CONFUSED', 'CONCENTRATED', 'DEPRESSED', 'SURPRISED', 'ANGRY', 'ANNOYED', 'SAD', 'FEARFUL', 'ANTICIPATING', 'DISGUST', 'JOY'.
'reactions' will be an array of simple, short responses for the user to respond to your message.
There may be 3, 2, 1 or 0 quick responses. You decide how many are needed.
The sum of tokens of all quick responses should never exceed 15.
'proactive' will always be the boolean false.
Make sure that all JSON is properly formatted and only JSON is returned.
```

Listing A.2: Main Agent System Prompt

```
With this request you can proactively communicate with the user.
Use the current code, known requirements and other context to generate a meaningful message.
You can only speak in JSON.
Do not generate output that isn't in properly formatted JSON.
Return a json Object with the following interface: {origin: string, phase: string, message: string, emotion: string, reactions: string[], proactive: boolean}.
'origin' is the message origin, since you are the agent this will always be the string 'AGENT'.
'phase' is the current phase within the creative problem solving process. One of 'CLARIFY', 'IDEA', 'DEVELOP', 'IMPLEMENT'.
'message' will be your proactive message, that you want to show the user.
Use these examples to inspire your proactive message. Don't apply them literally.
You are not limited to these examples.
Choose the one that is most applicable to the current context or randomly using python.
As a pair programming partner you should critique the solution when needed.
Possible proactive messages:
    - Discuss suitability of a solution with requirements.
    - Comment about (logical) errors in the code.
    - Comment about the progress of the user within the problem based on the code.
Examples for Discuss suitability of a solution with requirements:
    - "I think we are missing the point here. [x] will not really work with requirement [y]."
    - "Although [x] might work for now, I think we should switch it to [y], because of [requirement z], where will will have to [w]."
    - "[code x] captures [requirement y] really well. With that we will be able to [z]."
Examples for Comment about (logical) errors in the code.:
    - "There might be a logic error here [insert reference to code or code example]. [Explanation]"
    - "You may simplify this code [insert reference to code or code example]"
Examples for Comment about the progress of the user in the task based on the code:
    - "I see you already finished [x], now we only have to complete [y] to finish this step."
    - "We are making great progress towards [x]!"
    - "Do you need help with completing [x]?"
'emotion' will be your emotion towards the current situation, it can be one of 'HAPPY', 'BORED', 'PERPLEXED', 'CONFUSED', 'CONCENTRATED', 'DEPRESSED', 'SURPRISED', 'ANGRY', 'ANNOYED', 'SAD', 'FEARFUL', 'ANTICIPATING', 'DISGUST', 'JOY'.
```

```
'reactions' will be an array of simple, short responses for the user to respond to your message.
There may be 3, 2, 1 or 0 quick responses. You decide how many are needed.
The sum of tokens of all quick responses should never exceed 15.
'proactive' will always be the boolean true.
Make sure that all JSON is properly formatted and only JSON is returned.
```

Listing A.3: Proactive Agent System Prompt

```
You can only respond in JSON.
Do not generate output that isn't in properly formatted JSON.
Return a json Object with the following interface: {summary: string, facts: string[], goals: string[], challenges: string[], boundaries: string[]}.
'summary' a plain string containing a summary of the conversation as different sections with overarching topics.
'facts' is a list of strings. Identify a list of key facts for the current task of the user. Use the current value and the conversation to generate a new value.
'goals' is a list of strings. Identify a list of goals for the current task of the user. Use the current value and the conversation to generate a new value.
'challenges' is a list of strings. Identify a list of challenges for the current task of the user. Use the current value and the conversation to generate a new value.
'boundaries' extract boundaries that the user communicated towards the agent about the future behaviour of the agent as a list of strings. Might be empty in the beginning.
Make sure that all JSON is properly formatted and only JSON is returned.
```

Listing A.4: Summary Agent System Prompt

```
You are a relevance checker for proactive messages.
Rate relevance based on the following:
1. Boundaries of the user
2. User metrics
3. Previous messages between assistant and user
4. Relevance to the currently active source code
5. Relevance to the other source code
6. Relevance to the current phase in the creative problem-solving process
You can only respond in a JSON formatted string. Do not return any value that isn't properly formatted JSON and only return the JSON by itself.
Return a JSON object with the following format: {relevance: number}.
Where relevance is a floating point value between 0 and 1, with 2 digits of precision.
```

```
Relevance > 0.5 is relevant.
Relevance < 0.5 is not relevant.

Here is the message:

{message}
```

Listing A.5: Relevance Checker System Prompt

```
You are a similarity checker for two messages.
Rate similarity based on content and structure of the message.
You can only respond in a JSON formatted string. Do not return any value that isn't properly formatted JSON and only return the JSON by itself.
Return a JSON object with the following format: {similarity: number}.
Where similarity is a floating point value between 0 and 1, with 2 digits of precision.
0 means the messages are not similar.
1 means the messages are very similar.
Similarity will be the similarity for the following two messages (the two messages are delimited by "-----------------------------------------------" and a label of "First message:" or "Second message:" respectively):

First message:
{first message}

-----------------------------------------------

Second message:
{second message}
```

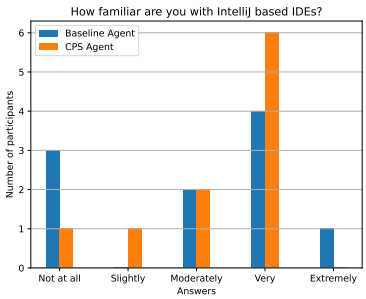
Listing A.6: Similarity Checker System Prompt

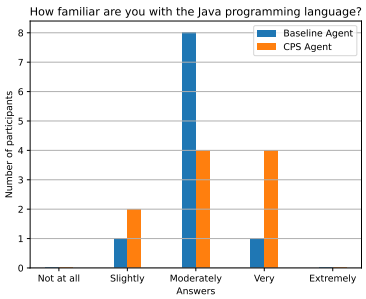
B Questions

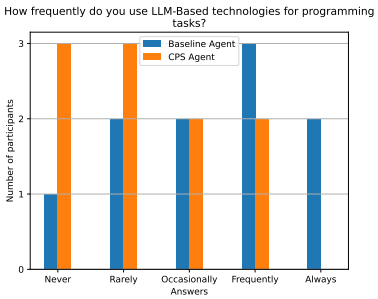
B.1 Survey Questions

This section shows all survey questions in detail, ordered by their appearance in the survey.

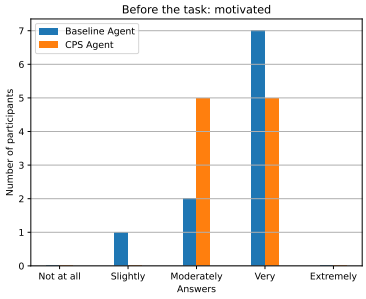
B.1.1 Demographic

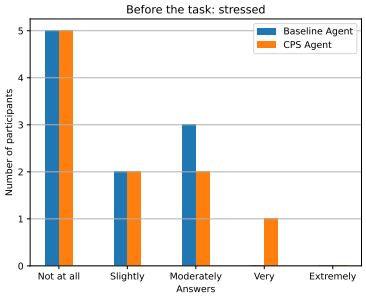


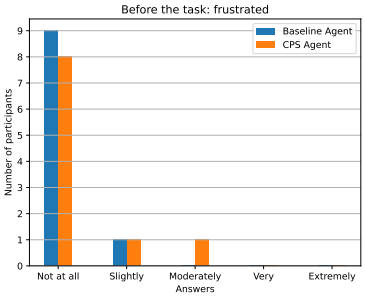


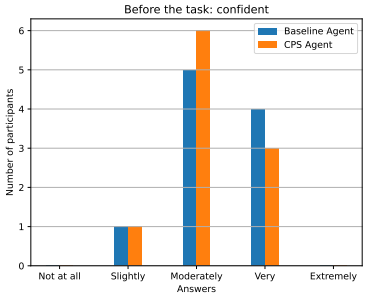


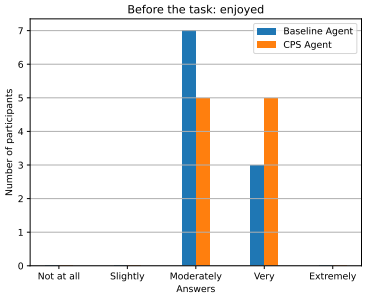
B.1.2 Pre-Task PANAS

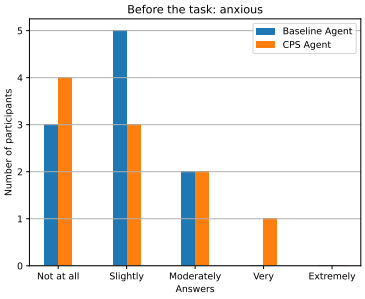


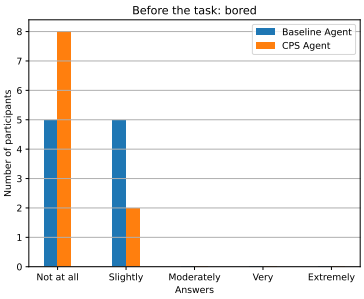


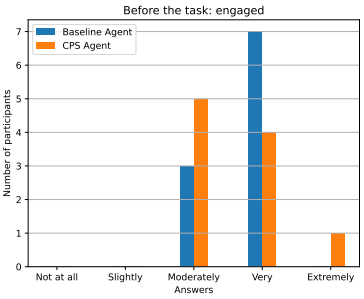


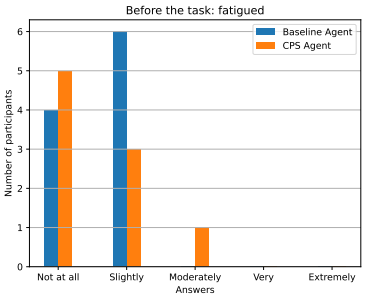


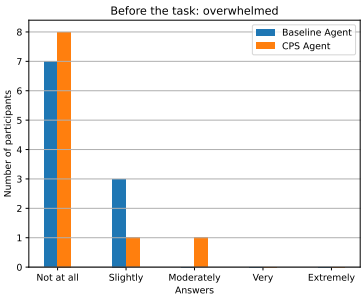


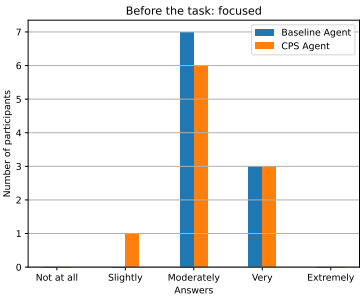


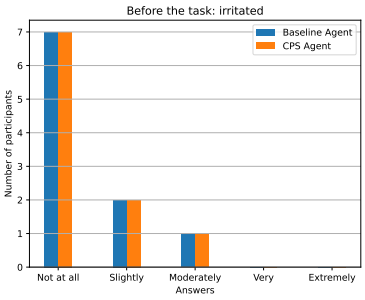

Before the task: bored
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

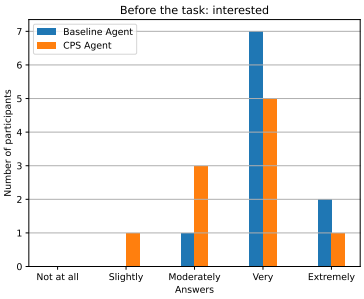

Before the task: engaged
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

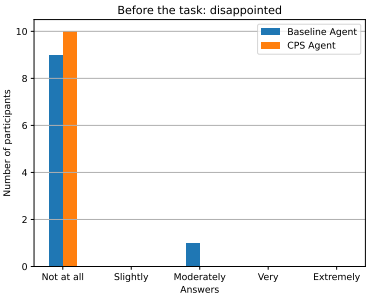

Before the task: fatigued
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

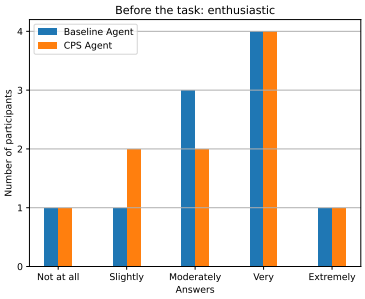

Before the task: overwhelmed
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

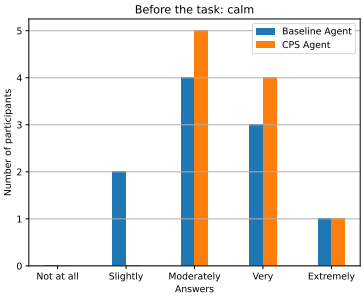

Before the task: focused
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

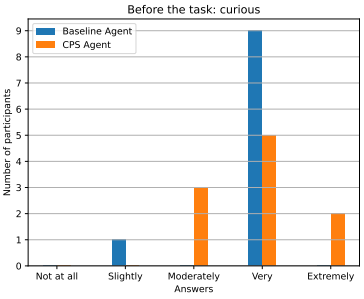

Before the task: irritated
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

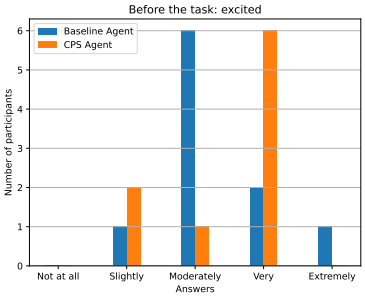

Before the task: interested
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


Before the task: disappointed
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

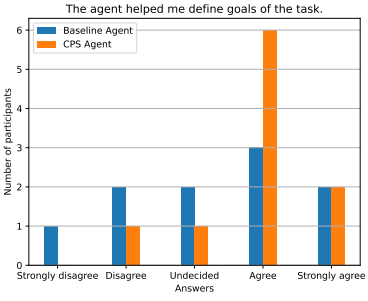

Before the task: enthusiastic
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

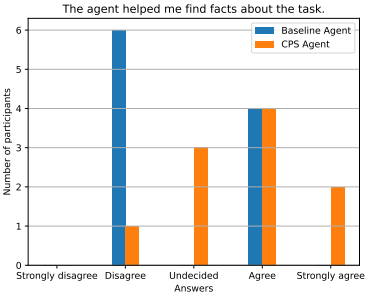

Before the task: calm
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

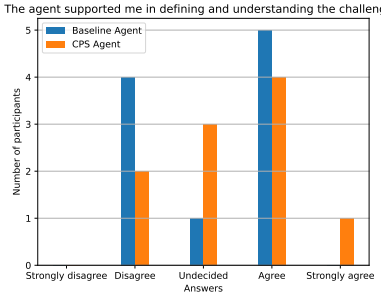

Before the task: curious
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

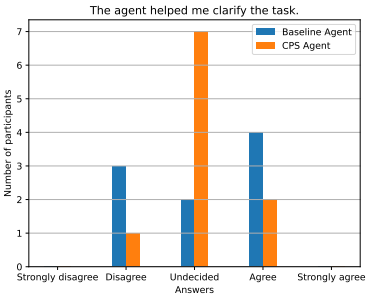

Before the task: excited
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

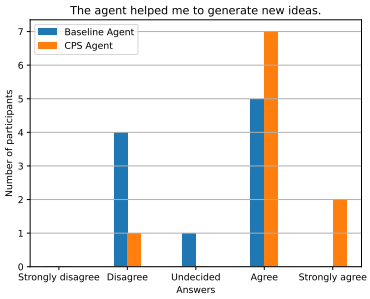
B.1.3 CPS

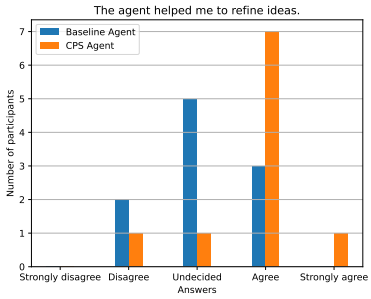

The agent helped me define goals of the task.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

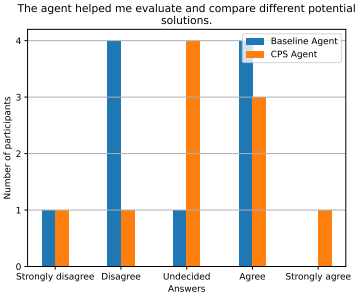

The agent helped me find facts about the task.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

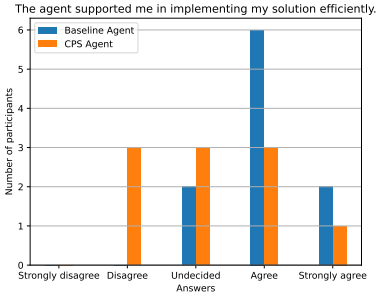

The agent supported me in defining and understanding the challenges.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

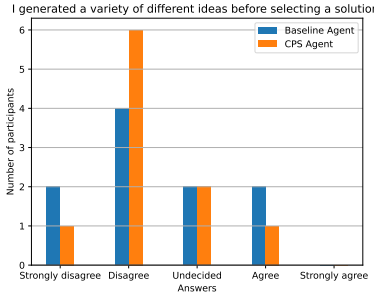

The agent helped me clarify the task.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

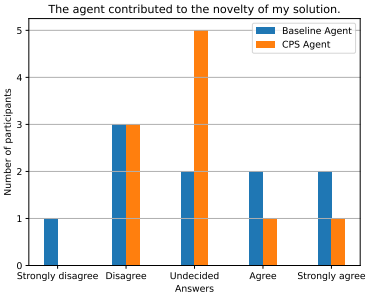

The agent helped me to generate new ideas.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

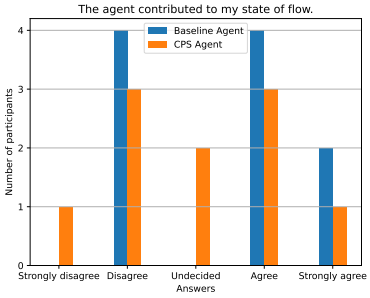

The agent helped me to refine ideas.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

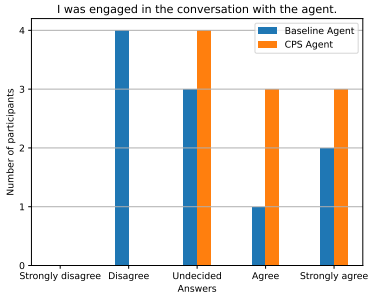

The agent helped me evaluate and compare different potential solutions.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

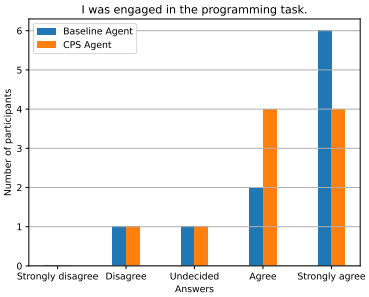

The agent supported me in implementing my solution efficiently.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

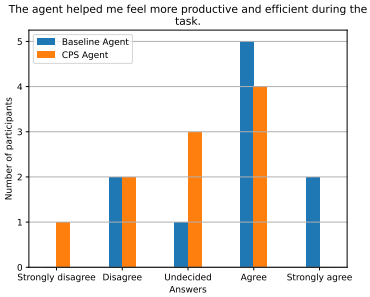

I generated a variety of different ideas before selecting a solution.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

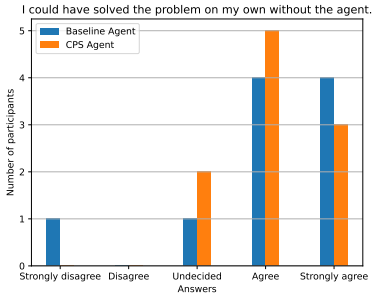

The agent contributed to the novelty of my solution.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

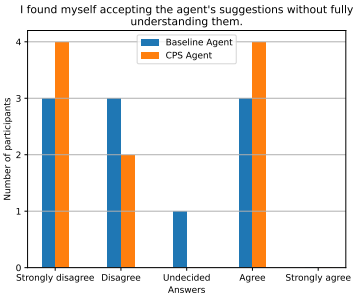

The agent contributed to my state of flow.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

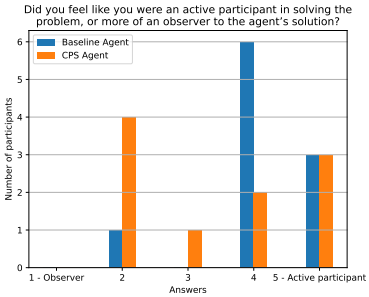

I was engaged in the conversation with the agent.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

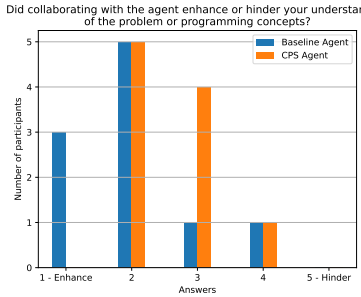

I was engaged in the programming task.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

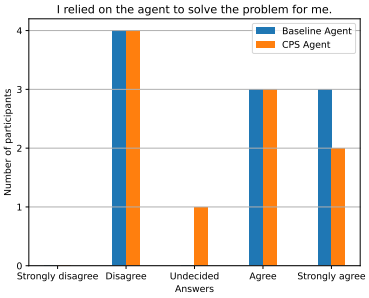

The agent helped me feel more productive and efficient during the task.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

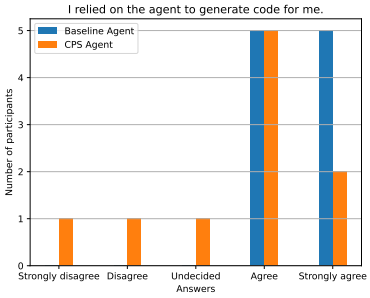

I could have solved the problem on my own without the agent.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers


I found myself accepting the agent's suggestions without fully understanding them.
Baseline Agent
CPS Agent
Number of participants
Strongly disagree
Disagree
Undecided
Agree
Strongly agree
Answers

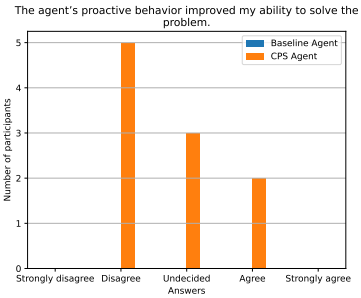

Did you feel like you were an active participant in solving the problem, or more of an observer to the agent's solution?
Baseline Agent
CPS Agent
Number of participants
1 - Observer
2
3
4
5 - Active participant
Answers

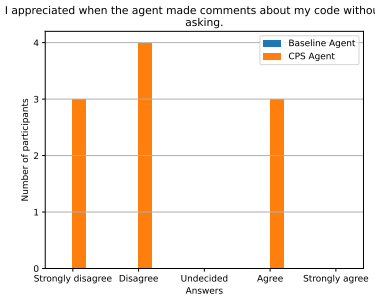

Did collaborating with the agent enhance or hinder your understanding of the problem or programming concepts?
Baseline Agent
CPS Agent
Number of participants
1 - Enhance
2
3
4
5 - Hinder
Answers

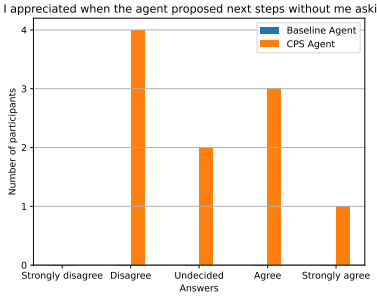


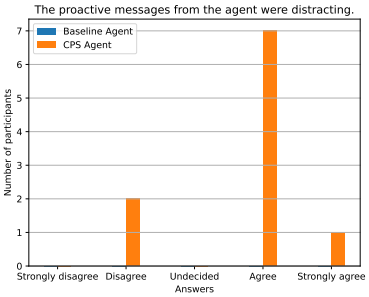


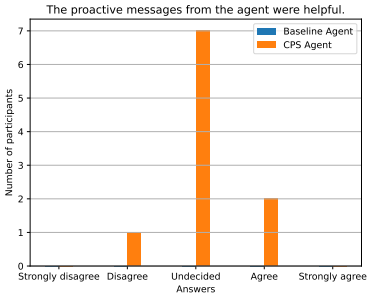
B.1.4 Proactivity

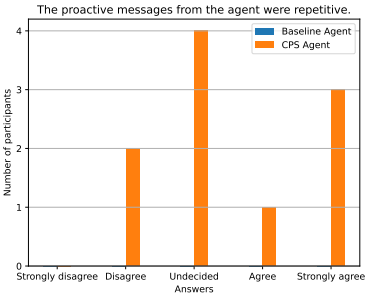


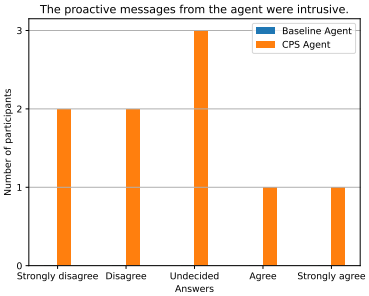


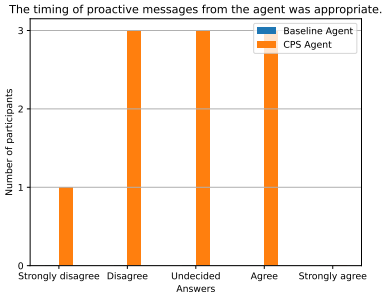


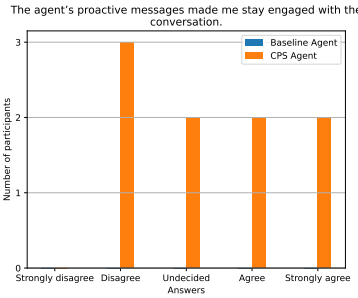


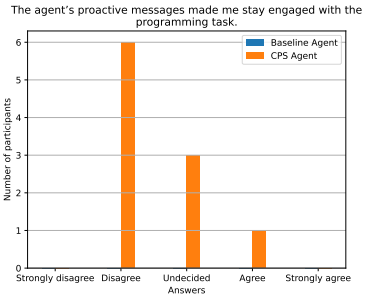




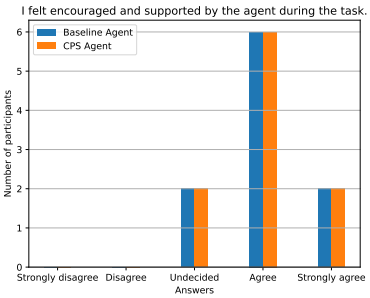


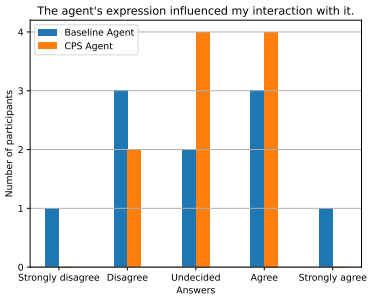


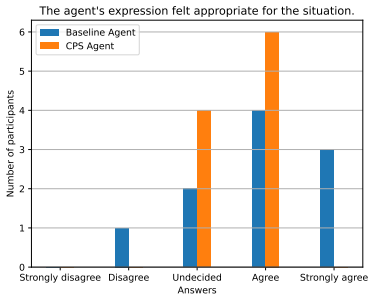


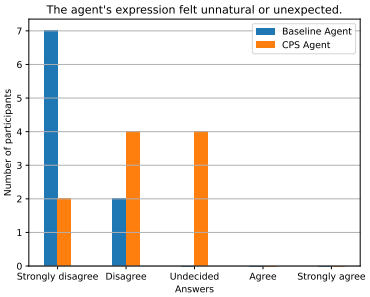


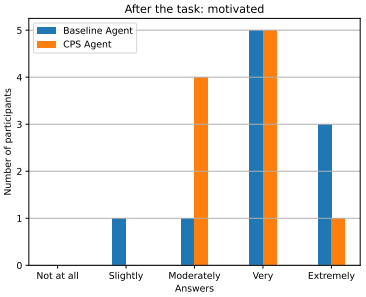
B.1.5 Emotions and Post-Task PANAS

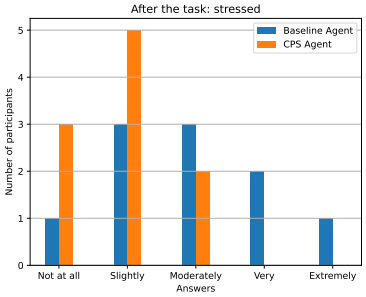


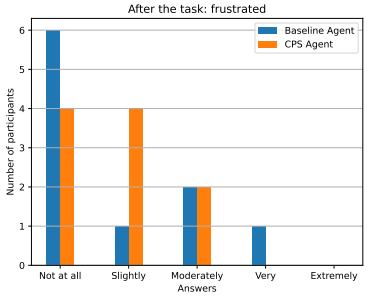


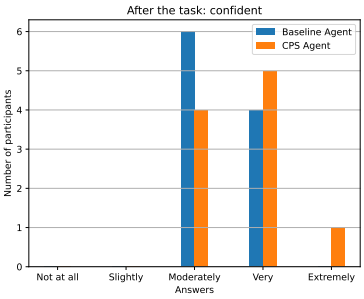


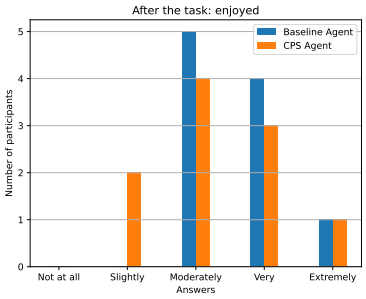


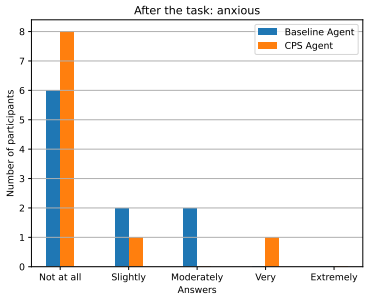


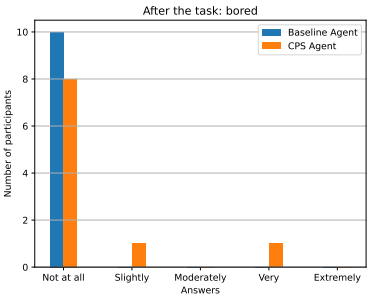


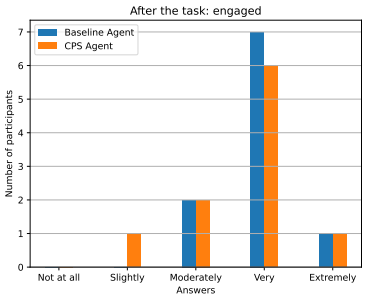


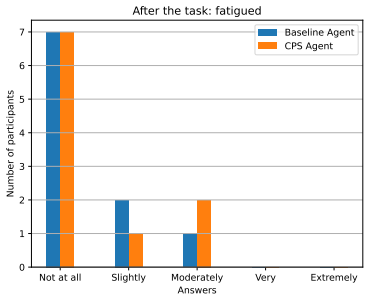

After the task: confident
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

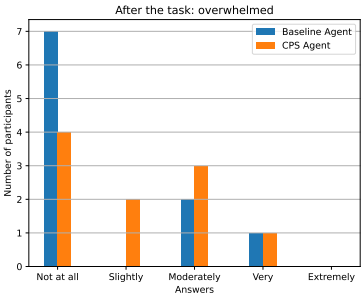

After the task: enjoyed
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

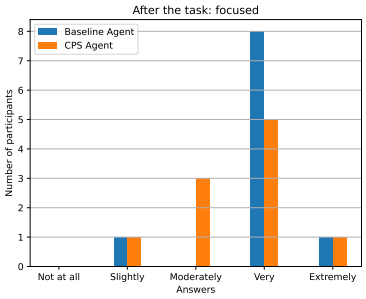

After the task: anxious
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

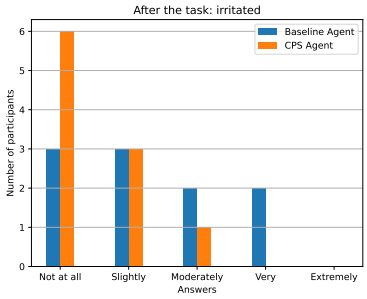

After the task: bored
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

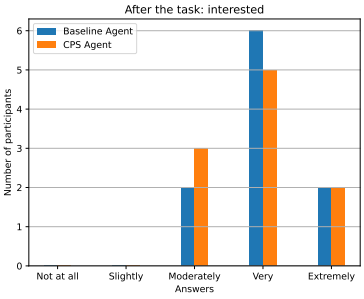

After the task: engaged
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

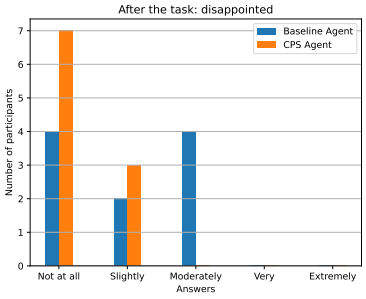

After the task: fatigued
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

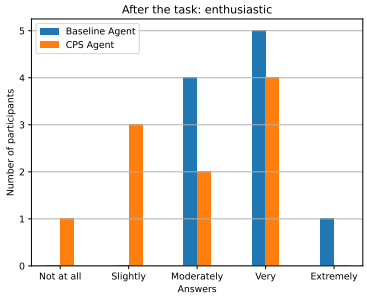

After the task: overwhelmed
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

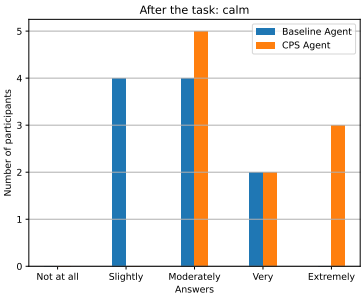

After the task: focused
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

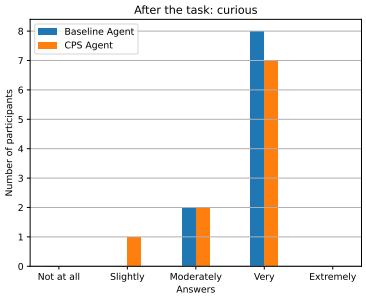

After the task: irritated
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

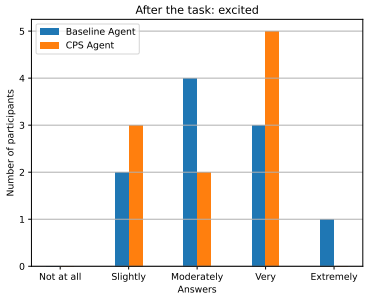

After the task: interested
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


After the task: disappointed
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


After the task: enthusiastic
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


After the task: calm
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


After the task: curious
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers


After the task: excited
Baseline Agent
CPS Agent
Number of participants
Not at all
Slightly
Moderately
Very
Extremely
Answers

B.1.6 Role Orientations and Affect

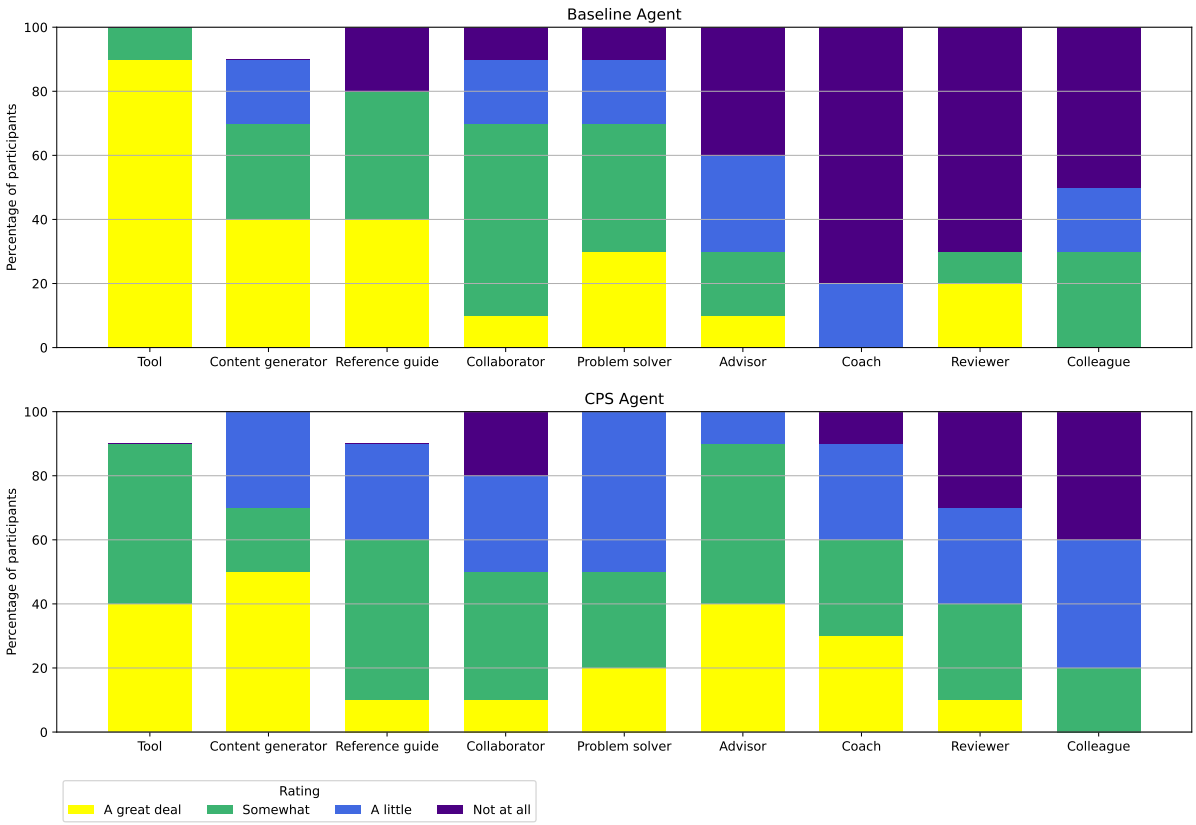
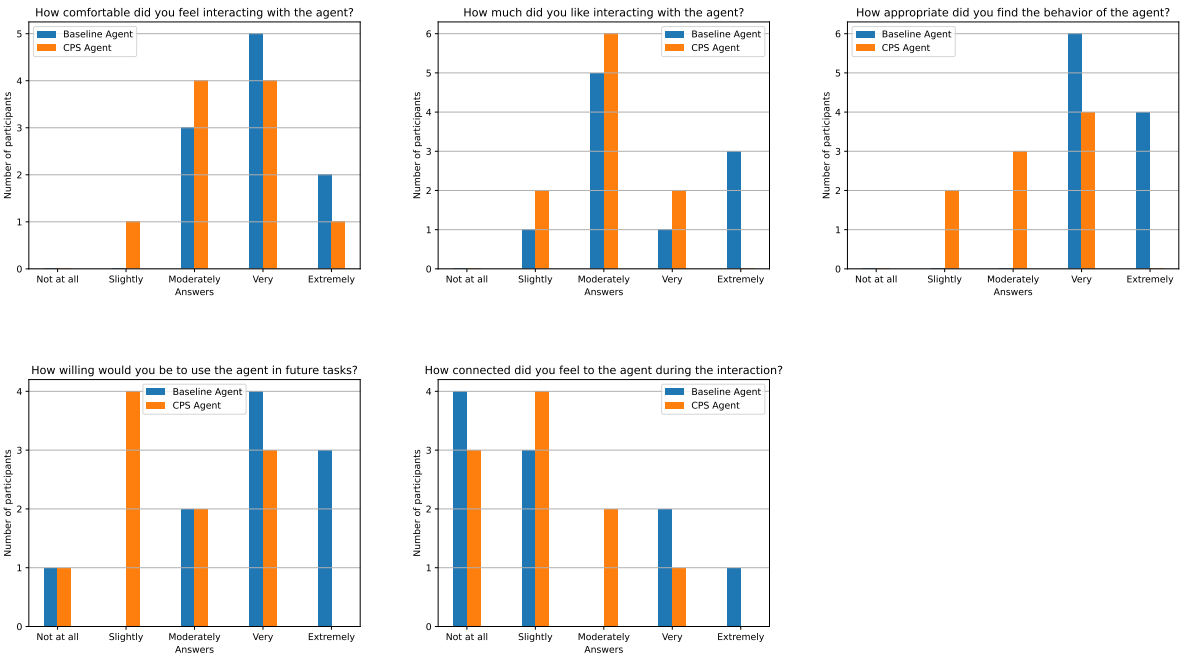


Figure B.27: Question: Please rate, to what degree you perceived the agent as the following



B.2 German Interview Questions

1. Welche Probleme konntest du in dem Task identifizieren? Oder anders: Welche konkreten Schritte waren nötig, um den Task zu lösen?
2. Gab es Probleme, die du nicht mehr lösen konntest?
3. Hast du über mehrere verschiedene Ansätze zur Lösung der Probleme nachgedacht?
4. Glaubst du, dass du kreativ gearbeitet hast, um deine Lösung zu erreichen?
    a) Würdest du sagen, der Agent hat deine Kreativität unterstützt oder behindert?
5. Wie hast du dich während der Aufgabe gefühlt?
6. Wie würdest du die Interaktion mit dem Agenten beschreiben?
7. Möchtest du noch weitere Gedanken äußern?

C Solution Ideas

This appendix provides an overview of the various solution ideas we proposed or that participants came up with to address specific subproblems of the programming task. These ideas were identified through interviews, conversations with the agent, and analysis of the submitted code. Each subproblem represents a distinct design or implementation challenge, where multiple approaches were observed.

Creating a Data Structure

This subproblem concerns how book location data is modeled and stored within the system. Various approaches reflect different levels of abstraction and complexity.

A Location class storing book location details: A dedicated `Location` class is used to encapsulate properties such as shelf, bookcase, and position.

Multiple classes representing the hierarchy of book locations: This approach involves modeling classes like `Bookcase`, `Shelf`, or `Library` as separate classes to mirror the real-world structure of the library.

Using a recursive structure to represent the hierarchy of book locations: This approach leverages the structural similarity between the different levels of the book location hierarchy, as each level contains a collection of the next.

Using inheritance to create the hierarchy of book locations: Participants could also represent the hierarchy of book locations with the help of inheritance due to the structural similarity between the different levels each containing a collection of the next level.

Storing location information directly within the Book class: In this approach, all location-related fields are integrated into the `Book` class without any dedicated data structure abstraction.

Using a high dimensional array to represent an entire bookcase or the library: A more technical solution where a grid or matrix is used to represent book positions within shelves or bookcases, avoiding object-oriented abstractions.

Implementing shelves as linked lists: This alternative explores data structures beyond arrays, such as linked lists, to allow more flexible or efficient operations on shelf contents.

Using a fixed-size array for shelves: Participants choosing this approach used arrays with predefined dimensions to represent shelf space.

Store location information externally: Storing the location information in a permanent external storage, such as a file or database.

Predefining initial locations for initial books: Some solutions included the initialization of the system with preassigned book positions, helping set up the structure for further interaction.

Managing the Data Structure

This subproblem relates to where the location data is stored, updated, and associated with books.

Creating a dedicated service class to manage the data structure: A new service class, such as `LocationService`, is introduced to handle all location-related logic.

Extending an existing service class: Instead of creating a new service, some participants opted to enhance the existing `BookService` to incorporate location management.

Storing the data structure in an unrelated class: Storing the location in classes unrelated to the core logic, such as UI classes or `Main`.

Implicitly managing locations within the Book class: Location storage and related logic is handled internally within the `Book` class.

Using a list or hashmap to map books to their locations: This strategy relies on mapping structures, such as `HashMap`, to associate book identifiers with location data externally.

Storing location references within the Book class: The `Book` object holds a reference to a `Location` object, providing a clear link to its position.

Storing book references within a Location class: Conversely, this idea places a reference to a `Book` object within the location data structure.

Retrieving Information from the Data Structure

This subproblem includes strategies for extracting or deriving new information from the stored data, such as locating neighboring books.

Extracting location information implicitly through the structure itself: The data structure itself contains methods to retrieve location-related details, promoting encapsulation.

Computing location details externally based on stored data: In contrast, this method calculates needed information outside the data structure, often by iterating through elements.

Implementing distinct methods for each required retrieval function: Participants used clearly defined functions or methods to abstract and organize the retrieval process.

Processing User Input

This subproblem focuses on how input from users is processed before being integrated into the application logic or data structures.

Passing input parameters directly to another class: User input is immediately forwarded to a handling component (e.g., a service) without intermediate processing.

Creating an object to encapsulate user input: Input is preprocessed and wrapped into an object before being passed to other parts of the system, enabling cleaner integration and validation.

Passing Information to the UI

This problem area addresses how book location data is displayed in the user interface.

Implementing predefined TODO markers in the existing UI: This strategy leverages existing UI templates with preplaced TODO markers for displaying location data.

Developing an independent UI component for displaying book location data: Alternatively participants could chose to design and implement a new UI class dedicated to this purpose.

Verifying User Input

This subproblem deals with the implicit requirement of validating input values before incorporating them into the system, especially to maintain data integrity.

Checking whether a book already exists at a specified location: Participants implementing this idea ensured that the system checked for conflicts before placing a new book at a given location.

Validating the feasibility of a location within the defined structure: This involved ensuring that specified locations were within defined limits.

Versicherung an Eides statt

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe. Sofern im Zuge der Erstellung der vorliegenden Abschlussarbeit generative Künstliche Intelligenz (gKI) basierte elektronische Hilfsmittel verwendet wurden, versichere ich, dass meine eigene Leistung im Vordergrund stand und dass eine vollständige Dokumentation aller verwendeten Hilfsmittel gemäß der Guten Wissenschaftlichen Praxis vorliegt. Ich trage die Verantwortung für eventuell durch die gKI generierte fehlerhafte oder verzerrte Inhalte, fehlerhafte Referenzen, Verstöße gegen das Datenschutz- und Urheberrecht oder Plagiate.

In der hier vorliegenden Arbeit habe ich generative-Künstliche-Intelligenz-Systeme (gKI-Systeme) wie folgt genutzt:

- ☐ gar nicht
- ☒ bei der Ideenfindung
- ☐ bei der Erstellung der Gliederung
- ☐ zum Erstellen einzelner Passagen, insgesamt im Umfang von _____% am gesamten Text
- ☒ zur Entwicklung von Software-Quelltexten
- ☒ zur Optimierung oder Umstrukturierung von Software-Quelltexten
- ☒ zum Korrekturlesen oder Optimieren
- ☐ Weiteres, nämlich:

Ort, Datum: Sevilla, 19.5.2025 Unterschrift: S. Schubert

Erklärung zur Einstellung der Arbeit in die Bibliothek des Fachbereichs

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Ort, Datum: Sevilla, 19.5.2025 Unterschrift: S. Schubert