Understanding Real-Time Collaborative Programming: A Study of Visual Studio Live Share

XIN TAN, School of Computer Science and Engineering, Beihang University, State Key Laboratory of Complex & Critical Software Environment (CCSE), Beijing, China

XINYUE LV, School of Software, Beihang University, Beijing, China

JING JIANG and LI ZHANG, School of Computer Science and Engineering, Beihang University, CCSE, Beijing, China

Real-time collaborative programming (RCP) entails developers working simultaneously, regardless of their geographic locations. RCP differs from traditional asynchronous online programming methods, such as Git or SVN, where developers work independently and update the codebase at separate times. Although various real-time code collaboration tools (e.g., *Visual Studio Live Share*, *Code with Me*, and *Replit*) have kept emerging in recent years, none of the existing studies explicitly focus on a deep understanding of the processes or experiences associated with RCP. To this end, we combine interviews and an e-mail survey with the users of *Visual Studio Live Share*, aiming to understand (i) the scenarios, (ii) the requirements, and (iii) the challenges when developers participate in RCP. We find that developers participate in RCP in 18 different scenarios belonging to six categories, e.g., *pair programming*, *group debugging*, and *code review*. However, existing users' attitudes toward the usefulness of the current RCP tools in these scenarios were significantly more negative than the expectations of potential users. As for the requirements, the most critical category is *live editing*, followed by the need for *sharing terminals* to enable hosts and guests to run commands and see the results, as well as *focusing and following*, which involves "following" the host's edit location and "focusing" the guests' attention on the host with a notification. Under these categories, we identify 17 requirements, but most of them are not well supported by current tools. In terms of challenges, we identify 19 challenges belonging to seven categories. The most severe category of challenges is *lagging* followed by *permissions and conflicts*. The above findings indicate that the current RCP tools and even collaborative environment need to be improved greatly and urgently. Based on these findings, we discuss the recommendations for different stakeholders, including practitioners, tool designers, and researchers.

CCS Concepts: • **Human-centered computing** → **Empirical studies in collaborative and social computing**; *Collaborative and social computing systems and tools*; • **Software and its engineering** → *Programming teams;*

Additional Key Words and Phrases: Real-time collaboration, software development, online collaboration, synchronous collaboration

1 INTRODUCTION

With the development of Internet technology, software development has become more flexible, allowing global developers to collaborate distributedly based on the Internet [6, 12, 24]. The popularity of this collaboration mode has been further accelerated due to the COVID-19 pandemic and is expected to continue to thrive in the long-term future [48]. In distributed software development, developers often work asynchronously when writing code. For example, multiple developers, designers, and team members work together on the same project based on version control systems (e.g., Git, CVS, and SVN) at separate times, which can keep track of file changes and allow multiple users to coordinate updates to that files [7, 54]. Although asynchronous programming collaboration is more convenient and common in distributed software development [42, 56], it does come with its problems. The serious problem is that asynchronous programming collaboration can create more space for miscommunication and bring latency, especially whenever there is a high degree of ambiguity, the topic is extensively detailed or complex, or the project requires immediate action [28].

Due to the intricacies involved in software development, especially in the context of large-scale systems, effective communication and collaboration among developers are essential [11, 51]. However, in distributed software development, where teams are geographically dispersed, traditional face-to-face collaboration becomes impractical. To address this challenge and meet the demands of developers in a highly competitive business environment, leading **integrated development environment (IDE)** manufacturers have introduced various IDEs or IDE plugins that enable **real-time collaborative programming (RCP)** [23, 36, 66]. Notable examples include *Microsoft*'s *Visual Studio Live Share*, released in April 2019, and *JetBrains*' *Code With Me*, released in April 2021.

In recent years, RCP also has garnered considerable attention from the academic community, as evidenced by the growing interest in this field [23, 36, 66]. Many studies have focused on investigating the advantages of RCP [10, 16, 63], as well as designing tools to address specific challenges associated with RCP in various contexts [23, 32].

Despite these efforts, there remains a gap in our comprehensive understanding of how developers engage in RCP. This includes various aspects such as the different scenarios in which RCP is employed, the specific requirements developers have when using RCP, and the challenges they encounter during the collaborative coding process. Gaining a deeper understanding of these aspects is crucial for effectively utilizing RCP and optimizing its potential benefits. To bridge this knowledge gap, we propose the following research questions.

  — RQ1: In what scenarios do developers participate in RCP?
  — RQ2: What are the requirements of developers when participating in RCP?
  — RQ3: What challenges do developers face when participating in RCP?

To answer the above questions, we conduct a two-step investigation with the users of *Visual Studio Live Share* (abbreviated to *Live Share*). First, to obtain a preliminary understanding of RCP, we conduct semi-structured interviews with 12 developers, each lasting about 50 minutes. Second, we surveyed 103 practitioners about their experiences with RCP to refine and generalize the interview findings. We find that developers participate in RCP in seven categories of scenarios. The top three common categories of scenarios are *pair programming*, *group debugging*, *interactive education*. Under these categories, we identify 18 scenarios (e.g., *breaking down tasks*), which

provide a more fine-grained understanding of application scenarios. During the RCP process, developers have various categories of requirements, e.g., *live editing* and *sharing terminals*. Under these categories, we identify 17 requirements, most of which are not satisfied by practitioners. As for challenges, we find 19 challenges (e.g., *different coding styles and ways of solution*) belonging to seven categories (e.g., *lagging*). Based on our findings, we propose a series of recommendations and areas for future research. This study takes the first steps in understanding the mechanisms of RCP, which can bring practical insights into the future optimization of RCP.

We organize the remainder of the article as follows. Section 2 presents background and related work; Section 3 introduces the methodology. Section 4 presents the results of each of the three research questions. Section 5 discusses our findings and illustrates the implications of findings for practitioners, tool designers, and researchers. Section 6 presents limitations, and Section 7 concludes the article. To replicate our study, we provide the interview script, survey questionnaire, codebook, and other relevant data in the replication package: https://doi.org/10.6084/m9.figshare.21834951.v4.

2 BACKGROUND AND RELATED WORK

RCP involves multiple developers working together on the same codebase simultaneously. It has gained increasing attention in both academia and industry in recent years due to its potential to enhance productivity, code quality, and knowledge sharing among team members. Most of these studies are in Software Engineering and **Computer-Supported Cooperative Work** (**CSCW**) and mainly focus on the benefits of RCP and tool support of RCP. In addition to research prototypes for RCP, top IDE manufacturers also released professional tools to support RCP.

2.1 Benefits of RCP

Through RCP, a group of developers can work together on the same code files and folders simultaneously, without having to manually resolve conflicts. This is done by instantly propagating, merging, and applying local editing operations at all collaborators' sites. Compared with Git-based workflows [55], RCP offers a lightweight collaboration process, allowing collaborators to easily initiate, join, and leave real-time collaboration sessions based on their changing needs. Studies [10, 16, 17, 63] have shown that RCP can accelerate problem-solving and collaborative innovation, similar to the benefits seen in synchronized collaboration in online learning [2] and engineering design [29]. It fosters frequent interaction among developers, setting it apart from loosely coupled and non-real-time collaboration (e.g., Git-based workflows). As a result, RCP is beneficial in various scenarios, including closely coupled collaboration in agile software development, distributed pair/party programming, online programming courses, and remote diagnoses and troubleshooting [35]. Overall, RCP boosts collaboration efficiency and effectiveness by promoting frequent interaction, accelerating problem-solving, and fostering collaborative innovation.

2.2 Tools for RCP

As shown in Table 1, both academia and industry have designed and implemented several tools for RCP. From a chronological perspective, researchers proposed the RCP prototypes as early as 2011. However, it was not until 2018 that commercial professional tools for RCP emerged.

In academia, there have been several prototypes of RCP tools. Examples include *Collabode* [23], *CoEclipse* [17], *CoRED* [32], *ATCoPE* [18], and *CoIDEA* [35]. These tools are specifically designed to address certain challenges in different scenarios. *Collabode* is a web-based Java-IDE that focuses on continuous compiling and debugging during RCP. It addresses the challenge of collaborative coding in the presence of program compilation errors introduced by other users and presents an algorithm for error-mediated integration of program code [23]. *CoEclipse* incorporates and

Table 1. Basic Information of RCP Tools

Type	RCP Tools	Created-by	Time	Characteristics	#Citations or #Downloads
Research prototypes	Collabode	Goldman et al.	2011	Web-based Java IDE focusing on continuous compiling and debugging during RCP	169 [23]
	CoEclipse	Fan et al.	2012	Eclipse-based IDE achieving integrated consistency maintenance and awareness in RCP environments	23 [17]
	CoRED	Lautamäki et al.	2012	Browser-based RCP editor for Java applications with error-checking and automatic code-generation capabilities	30 [32]
	ATCoPE	Fan et al.	2012	Eclipse-based IDE aiming to integrate RCP with Non-RCP	24 [18]
	CoIDEA	Ma et al.	2023	Eclipse-based IDE allowing programmers to closely collaborate in a real-time fashion while preserving the work's compatibility with traditional non-real-time collaboration	2 [35]
Professional tools	Live Share	Microsoft	2019	Visual Studio Code Plugin allowing users to co-edit, co-debug, chat with peers, share terminals/servers, look at comments, and so on.	14M+ [38]
	Code With Me	JetBrains	2021	Standalone IDE aiming to empower the developer team to invite members into their IDE project and collaborate on it in real time	580K+ [37]
	Replit	Replit	2018	Web-based IDE enabling collaborators to run code live in their browsers without local configuration	22M+ [5]

Except for the statistics data provided by Replit, which is only available until April 2023, the data from other tools is up-to-date as of November 2023.

The numbers in the last column of research prototypes (professional tools) are number of citations (downloads).

integrates various techniques derived from continuous research work in the domain of RCP to achieve integrated consistency maintenance and awareness in RCP environments [17]. *CoRED* is a browser-based RCP editor for Java applications with error-checking and automatic code-generation capabilities, as well as some features commonly associated with social media [32]. *AT-CoPE* aims to seamlessly integrate conventional non-RCP tools and environments with emerging RCP techniques and support collaborating programmers to work in and flexibly switch among different collaboration modes according to their needs [18]. *CoIDEA* integrates real-time and non-RCP with a novel workflow and contributes enabling techniques to realize such integration [35]. These prototypes demonstrate the ongoing research and innovation in the field of RCP, catering to specific challenges and offering diverse integration possibilities for effective collaboration.

There are already some mature real-time collaboration tools, such as Google Meet[1] and zoom[2]. However, these tools only support video/audio conferences, which makes it hard to meet the diverse demands of developers [13]. To better support RCP, top IDE manufacturers are producing various tools. Among these tools, the most typical ones include *Live Share*[3], *Code With Me*[4], and *Replit*[5]. The main difference between these emerging tools and traditional RCP tools is that these emerging tools provide an IDE allowing multiple developers to browse and edit the same code files simultaneously. Therefore, these tools truly realize real-time code collaboration between developers. Then, we briefly introduce these tools (please refer to Table 1) and explain why we chose *Live Share* as our study case.

Live Share is an extension for *Visual Studio*, which is released by Microsoft in April, 2019. It allows users to co-edit, co-debug, chat with peers, share terminals and servers, look at comments, and so on. It is prevalent and has been installed 14M+ times until November 2023 [38]. *Code With Me* is a collaborative development service released by JetBrains in April 2021. Similar to the features provided by *Live Share*, *Code With Me* is specially designed to empower the developer team to invite members into their IDE project and collaborate on it in real time. The major differences compared

---

[1]https://meet.google.com/

[2]https://zoom.us/

[3]https://visualstudio.microsoft.com/services/live-share/

[4]https://www.jetbrains.com/code-with-me/

[5]https://replit.com/

Semi-structured
Interview

Designing
Questionnaire

Data
Analysis

Designing
Interview Script

Qualitative
Analysis

Sending
Questionnaire

Results

Phase I − Semi-structured Interview
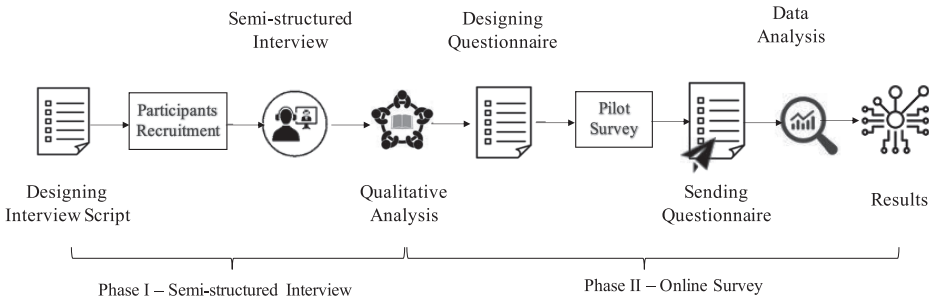
Phase II − Online Survey

Fig. 1. Overview of the research design.

with *Live Share* are the built-in support for video calls, the availability of the on-premises solution for customers who need it, and potentially the refactoring functionality [3]. Until November 2023, it has been downloaded 580K+ times [37]. Different from the above two tools, *Replit* is the world-leading collaborative browser-based IDE, which means that collaborators can run code live in their browsers without local configuration. Except for this difference, its collaboration features are similar to the two tools mentioned above. It has 22M+ users around the world until April 2023 [5].

Despite existing studies and professional tools that have been developed, none of the existing studies explicitly focus on a deep understanding of the underlying processes or experiences associated with RCP. This knowledge gap hinders the optimization of this collaborative mechanism and the improvement of RCP tools. To address this gap, our study focuses on investigating RCP through Microsoft's *Live Share*. We choose *Live Share* for several reasons. (1) As an RCP tool launched by Microsoft, the world's top IT enterprise, *Live Share* has all the core functions of mainstream RCP tools. (2) *Live Share* has accumulated substantial users since its first release in 2019. Out of the above two reasons, investigation of its users can help us understand the current situation of RCP more objectively. (3) *Live Share* hosts its issue tracker on GitHub. We can get the public e-mail addresses of its users through GitHub user profiles, allowing us to interact with the developers with RCP experience.

3 METHODOLOGY

We devised a study including two phases, as depicted in Figure 1. Due to a lack of prior knowledge of RCP, we first conducted a small-scale interview to gather insights and firsthand experiences from participants who have engaged in RCP. Specifically, through the interviews, we aimed to preliminarily understand their scenarios, requirements, and challenges. The information obtained through the interviews serves as the basis for designing the questionnaire for the following survey. The subsequent survey was designed to validate and enrich the findings gained from the interviews. By reaching a larger sample of participants, the survey can provide a broader perspective compared with the interviews. Additionally, the survey allowed us to gather quantitative data to support and strengthen the qualitative insights. The ethical committee of Beihang University approved the research protocol. We provide the approval letter in the online appendix.

3.1 Semi-Structured Interview

We first introduce how we design the interview script, then introduce how we recruit developers to participate in the interviews, and finally introduce how we analyze the interview results.

3.1.1 Designing Interview Script. To the best of our knowledge, no studies specifically focus on RCP. To get a primary understanding of RCP, we carefully read the official documents of *Live*

Table 2. Structure of Semi-Structured Interview

Section	Question/Content
A. Introduction	Explaining research objectives; Remembering the terms of the consent form; Asking permission for recording.
B. Overview	Brief description of current job.
C. Experience	Motivations for participating in RCP. Aiming to obtain scenarios; Frequency of participating in RCP.
D. Requirements & Challenges	General Aspect • Core demands of RCP; • The biggest challenges faced when conducting RCP; • RCP tools used and evaluation. Specific Aspect • Importance/ Challenges/ Optimization of "live editing"; • Importance/ Challenges/ Optimization of "focus and follow"; • Importance/ Challenges/ Optimization of "chat"; • Importance/ Challenges/ Optimization of "group debugging"; • Importance/ Challenges/ Optimization of "sharing servers"; • Importance/ Challenges/ Optimization of "sharing terminals".
E. Wrap Up	Anything else would like to tell.

Share and focused on its usage scenarios and main features. Then, all the authors attempted the most popular RCP tools (i.e., *Live Share*, *Code With Me*, and *Replit*) together and tried to have an idea as to the features of each. At this point, we knew what activities developers might conduct during the RCP process. After finishing the above process, we started brainstorming potential interview questions relating to the three research questions concerning scenarios, requirements, and challenges of RCP. Please note that although some specific codes (e.g., scenarios in Table 5) have been mentioned in the official documents, we deliberately avoided explicitly mentioning such fine-granularity questions during the interviews to prevent any biases or preconceptions.

We followed the suggestions of Newcomer et al. [43] to design our semi-structured interview. For example, in terms of the order of questions, we usually started with general questions and then proceeded into detail to avoid our subjective thoughts interfering with the interviewees' answers. We also adjusted our questions according to the interviewees' answers. For example, we asked about possible optimizations only when the interviewees mentioned challenges. Table 2 presents the structure of our interview script. The interview is divided into five parts, among which Parts C&D contain the main content of our interview. For the "specific aspects" in Part D, we mainly want to obtain their opinions on the main functions of these RCP tools (e.g., *live editing*) to understand their requirements. For a detailed explanation of these functions, please refer to Section 4.2.

3.1.2 Recruiting Participants and Conducting Interviews. To recruit interviewees, we focus on the developers who have participated in the issue discussion about *Live Share* on GitHub[6]. We assume that these developers all have experience in RCP. By August 2023, there were 4,751 issues reported and 3,900 developers participated in the issue discussion. However, not all developers publish their email addresses on GitHub. Eventually, we obtained 1,278 e-mail addresses through GitHub API. To avoid disturbing too many developers, we randomly selected 200 developers and

---

[6]https://github.com/MicrosoftDocs/live-share/issues

Table 3. Demographics of Interviewees

ID	Gender	#Years of PRO EXP in SE	Location	Occupation	Frequency of Participating in RCP
P1	male	10	China	PhD student	sometimes
P2	male	2	Ghana	software engineer	sometimes, but every day for the past period of time
P3	male	9	Thailand	founder of online learning community	every week
P4	male	6	Netherlands	open source developer	sometimes
P5	male	5	Ghana	software engineer	very often for the past period of time
P6	male	8	Sweden	consultant working for government	every day
P7	male	6	Kenya	software engineer	every day
P8	male	1	India	software development intern	every week
P9	male	3	France	software engineer	sometimes
P10	male	3	China	software engineer	very often for the past period of time
P11	female	8	China	software engineer	every week
P12	female	4	China	master student	very often for the past period of time

"For the past period of time" means that developers have been engaged in a software development project involving RCP.

sent our interview invitations through e-mail. Twelve developers showed their interest in our interview; finally, ten successfully completed the interview. Since all ten interviewees were male, we took the initiative to diversify the pool by reaching out to our own social network and recruiting two female interviewees who have experience with RCP to participate in the interviews. Table 3 shows the demographics of the interviewees. The first two authors together interviewed 12 participants in turn through Google Meet[7], and each interview lasted about 50 minutes. To express our gratitude, we sent a reward of 50 USD after each interview.

3.1.3 Analyzing Interview Results. We first transcribed the audio data (625 minutes in total) into the text data. Then, we performed open card sorting [67] to extract emerging codes and themes. The codes were organized into hierarchies through this process to deduce a higher level of data abstraction. Specifically, the first two authors created self-contained units (i.e., codes) and then sorted them into themes. For instance, an interviewee said that *There are also some terminals that do not support any colors. For example, on my side, everything appears normal, but others are unable to see any colors.* Based on this feedback, we identified the code "cannot adapt to the configuration of other PCs". Consequently, we incorporated this code into the broader theme of "adaption to other tools/plugins". After this process, the first two authors interactively sorted the codes several times to ensure the themes' integrity and validity. Then, they discussed the emergent codes and themes in weekly hands-on meetings to reach a negotiated agreement [21]. After the third author's review and a series of discussions, we reached the final themes. During the discussions, we examined controversial codes and reduced potential bias induced by incorrect interpretations of a participant's answer. Eventually, we obtained 48 codes belonging to 19 themes[8], which are

---

[7]https://meet.google.com/

[8]It should be noted that besides the interview transcripts, seven codes related to the "scenario" were obtained from *Live Share* official documents. We validated these codes through the latter online survey.

Table 4. Examples of the Survey Questions

q11	Below are the requirements that you may have when participating in RCP. Please indicate the extent to which you agree with each of the following requirements. (*Strongly disagree*, *disagree*, *Neither agree nor disagree*, *Agree*, *Strongly agree*, *Don't know/no opinion*)
q12	Do you have other requirements when you participate in RCP that are not listed above? If so, please list them here.

available in our replication package. We do not share the interview transcripts due to confidentiality reasons. However, we made our complete code book publicly available within the supplemental material. The code book includes all theme names, code names, and examples of quotes.

3.2 Online Survey

First, we introduce how we design the questionnaire. Second, we introduce how we recruit participants and conduct an online survey. Last, we introduce how we analyze the survey results.

3.2.1 Designing Questionnaire. In phase II of our study, we surveyed developers who have ever participated in RCP. We aims to validate and expand on the findings collected through the semi-structured interviews and answer the three research questions. The survey has 22 questions that are divided into seven parts. Most of the options are obtained through semi-structured interviews. To avoid the "order-bias" [47], we randomized the order of options for the multiple-choice questions. Except for one qualification question, developers were allowed to skip any questions if they were unwilling to answer. Table 4 shows an example of a closed and an open-ended question. In the following, we explain the design of each part of the questionnaire. The complete survey is available in the replication package.

(1) Introduction and Qualification. On the first page, we inform participants about the information of our survey, e.g., the purpose, number of questions, and data handling policy. We also inform them that they can drop out at any time. We set up a qualification question to ensure that all the participants are qualified and understand our policies. This question has three options, i.e., A. *have read the consent form*, B. *agree to participate*, and C. *have experience in RCP*. Only when participants select all three options can they proceed to the next part of the questionnaire.

(2) General Experiences of RCP. In the second part, we want to know developers' general experiences in RCP, so we asked the following four questions. First, we ask respondents about their use of mainstream RCP tools, i.e., *have used*, *not have used but heard*, and *never heard*. Second, we ask respondents how they know these tools. We provide several options, e.g., *job need*, *personal interest*, and *recommendation*. Then, we ask respondents how often they participated in RCP in the past 12 months. In the last question, we ask respondents to evaluate the usability [20] of the RCP tools that they use most often. We set three 5-point Likert scale questions from effectiveness, efficiency, and satisfaction, respectively. The corresponding final scores are 4.12, 4.17, and 3.89.

(3) Scenarios. This part aims to obtain the scenarios in which developers participate in RCP. We set four questions. First, we ask respondents to select the three most common scenarios, for which we provide some options based on our interview and official documents of *Live Share*, e.g., *pair programming*, *interactive education*, and *code review*. Second, we wonder where developers most often participate in RCP, i.e., at home, company, or in other places. Third, to obtain a deeper understanding, we provide a detailed list of scenarios and let respondents check whether they have participated in these scenarios and whether they think RCP is/may be helpful in these scenarios. In the end, we set an open-ended question to let respondents add other scenarios not mentioned.

(4) Requirements. In the fourth part, we want to know the requirements of developers when participating in RCP. We set three questions. Similar to the questions in the previous part, we

first ask respondents to select the three most essential requirements, e.g., *live editing*, *focus and following*, and *text chat*. Then, we provide a 5-point Likert-scale grid that contains 12 detailed requirements, for which we ask the extent to which they agree. We also set an open-ended question to let respondents add other requirements not mentioned.

(5) Challenges. In addition to the scenarios and requirements, we also want to know the challenges developers may face while participating in RCP. Similar to the above two parts, we set three questions. First, we asked respondents to select the three most serious challenges. We provide several options, e.g., *lagging*, *adaption to different machines*, and *safety and privacy*. Then, we provide a 5-point Likert-scale grid that contains 19 detailed challenges and ask respondents to rate their agreement. For the last question, we set an open-ended question to let respondents add other challenges not mentioned.

(6) Demographic. We ask respondents questions to collect demographic information and confounding factors, i.e., *occupation*, *professional experience in software engineering*, *company size*, and *location*.

(7) Final Feedback. In the last part of our survey, we set three open-ended questions. First, we ask developers' opinions on the differences between RCP tools (e.g., *Live Share*), non-RCP tools (e.g., *GitHub*), and online meeting tools (e.g., *Google meet*). Then, we hope to receive any comments on our survey. In the end, we ask respondents whether they would like to receive the survey results.

3.2.2 Recruiting Participants and Conducting Online Survey. We built and ran the survey on Google Forms[9]. Before conducting the formal survey, we conducted a pilot survey [60] to test our questionnaire. We invited three participants (i.e., a professor, a graduate student in computer science, and a developer from an IT company) who have RCP experiences to get suggestions for the questionnaire design. The suggestions we obtained are mainly related to the wording of the questions and the setting of options, e.g., for each Likert-scale question, add an option *I don't know* to leave a back door for those who cannot make up their mind. To recruit participants, we e-mailed survey invitations to 1,278 developers referenced in Section 3.1.2 in August 2023. These developers had previously engaged in issue discussions for *Live Share* on GitHub and had publicly shared their e-mail addresses. Ultimately, 1,151 e-mails were successfully delivered. To enhance survey participation, we implemented various strategies, including personalized invitations and maintaining participant anonymity [53]. After three weeks, we received 66 responses. In addition to the aforementioned questionnaires, we also utilized the authors' social connections in November 2023, leveraging platforms such as WeChat circle of friends and WeChat groups to distribute the survey. Through this approach, we obtained 55 responses. Subsequently, the responses underwent manual screening for anti-patterns (e.g., all responses being empty or featuring identical values for all questions). Eighteen responses were consequently excluded. As a result, we ultimately obtained 103 valid responses.

Figure 2 shows the demographic of the participants. Half of the participants come from China while the others come from all over the world. These participants have different experiences in software development. Almost all of them have used *Live Share* and participated in RCP several times in the past 12 months. Among these respondents, 49% and 12% participate in RCP weekly and monthly. Forty-eight percent of the respondents participate in RCP at home. It indicates that the physical isolation of the team caused by work-from-home is an essential factor in the occurrence of RCP. To express our gratitude for the respondents' precious effort and contribution, we donated 5 USD per response to an OSS project selected by the respondents.
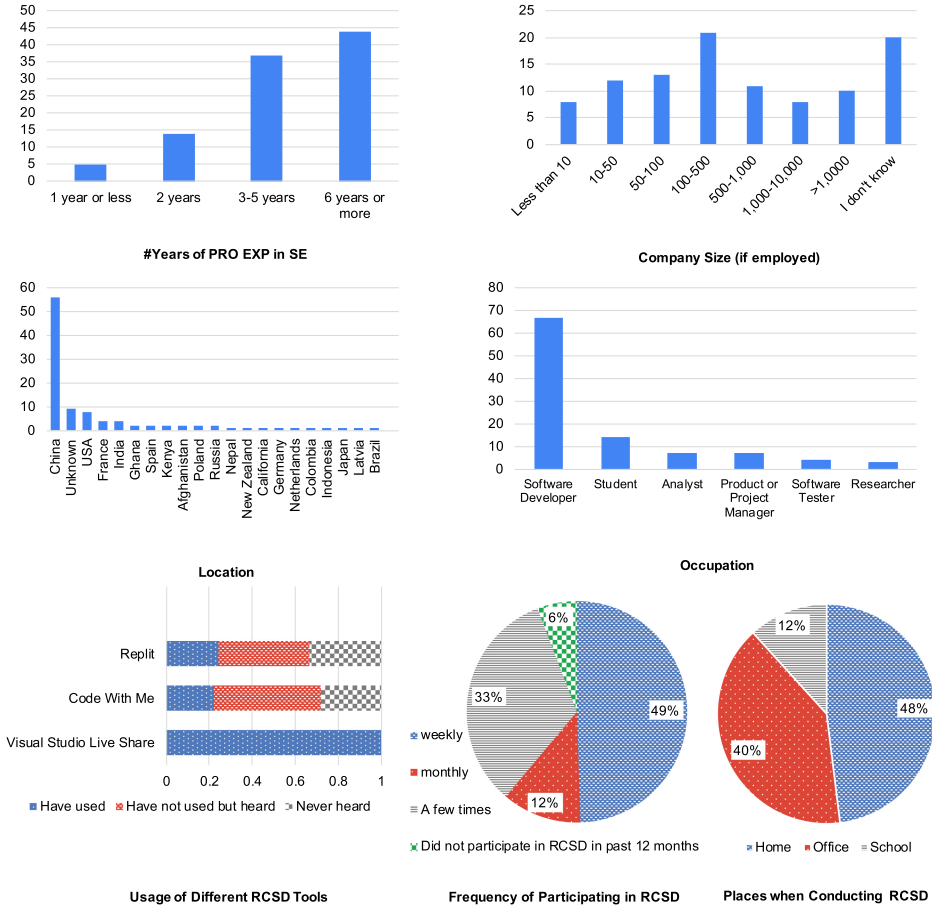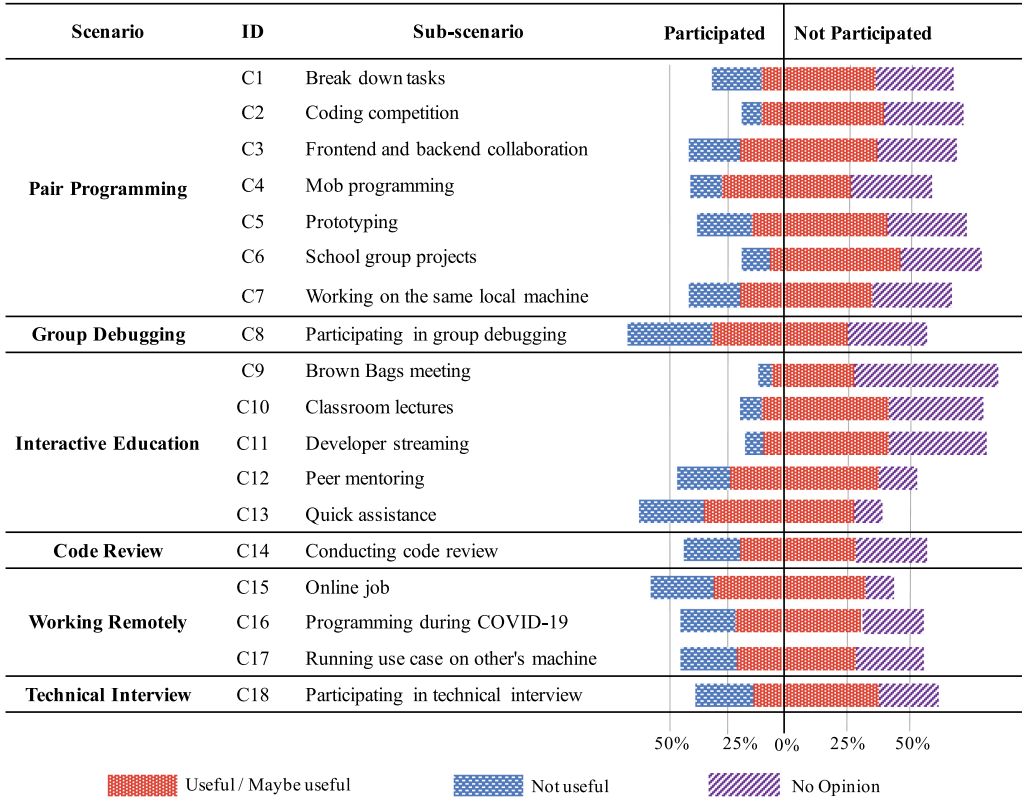
---

[9]https://www.google.com/forms/

Fig. 2. Demographics of survey respondents.

3.2.3 Analyzing Survey Results. We followed the suggestions of Chambers et al. [8] to analyze the survey results. For the close-ended questions, we reported the distribution and statistical analysis results. For the results of Likert-scale questions, we referred to Petrillo et al. [46] analysis method to compute several statistics to evaluate developers' attitudes toward given statements. Firstly, we converted *strongly disagree*, *disagree*, *neutral*, *agree*, *strongly agree* to 1, 2, 3, 4, and 5, respectively. We then computed the median for all the scores. For ratings of statements, we calculated the percentage of respondents who strongly agree or agree with each statement (% strongly_agree + % agree) as "% Agreement". By referring to previous studies [59, 62], we also computed a conflict factor for each statement. The conflict factor is a measure of disagreement between respondents, which is calculated for each statement by the following equation: $abs(\frac{(strongly\_agree+agree)-(strongly\_disagree+disagree)}{strongly\_agree+agree+strongly\_disagree+disagree})$. Theoretically, the range of a conflict factor is [0, 1]. When the value is equal to 0, the disagreement between respondents is the largest, and when the value is closer to 1, the disagreement between respondents is smaller. For the open-ended questions, we adopted the open card sorting process similar to Section 3.1.3. Through this process, we obtained six codes and one theme that were not discovered in our semi-structured interview.

Table 5. Scenarios of RCP. The Bar Chart Represents Developers' Opinions Obtained by Online Survey

Scenario	ID	Sub-scenario	Participated	Not Participated
Pair Programming	C1	Break down tasks		
	C2	Coding competition		
	C3	Frontend and backend collaboration		
	C4	Mob programming		
	C5	Prototyping		
	C6	School group projects		
	C7	Working on the same local machine		
Group Debugging	C8	Participating in group debugging		
Interactive Education	C9	Brown Bags meeting		
	C10	Classroom lectures		
	C11	Developer streaming		
	C12	Peer mentoring		
	C13	Quick assistance		
Code Review	C14	Conducting code review		
Working Remotely	C15	Online job		
	C16	Programming during COVID-19		
	C17	Running use case on other's machine		
Technical Interview	C18	Participating in technical interview		

50% 25% 0% 25% 50%

Useful / Maybe useful   Not useful   No Opinion

4 RESULTS

In this section, we report the results of our investigation based on semi-structured interviews and an online survey. Eventually, we obtained 54 codes (interviews: 48 codes; survey: 6 codes) and 20 themes (interviews: 19 themes; survey: 1 theme) related to the three research questions. We quote some of the interviewees' words to illustrate our findings more clearly.

4.1 RQ1: Scenarios

As shown in Table 5, we find that developers participate in RCP in 18 different scenarios, among which 11 scenarios were derived from the interviews, seven scenarios were derived from the Live Share official documents, and we did not obtain any new scenarios from the survey. On average, a single respondent participated in six scenarios. All the 18 scenarios belong to six categories. We first ask developers to choose the most common categories of scenarios they have participated in. Among them, **Pair Programming** and **Group Debugging** are the top two most common categories of scenarios, while **Code Review** is in third place. Then, for each scenario, we investigate its frequency and the helpfulness of RCP in this scenario. If developers have not participated in a certain scenario, we asked how much they agreed or disagreed that RCP might be helpful in this scenario. Although an average of 53% of respondents believe that RCP is / may be useful in certain scenarios, there is a significant difference in attitude between those who have actually participated in these scenarios and those who have not (Mann–Whitney U Test [41]: The

z-score is −3.65. The *p*-value is < .001. Overall, 58% of respondents who have not participated in specific scenarios believe that RCP may be useful, while 45% of respondents who have participated in specific scenarios believe that RCP is useful. It indicates that there is still room for improvement in the current RCP tools compared with developers' expectations, reflecting the significance and necessity of exploring developers' requirements (RQ2) and challenges (RQ3) in conducting RCP.

(1) Pair Programming. This is the most common category of scenarios for RCP. As the name implies, pair programming is a practice in software development where two developers collaborate in a single workstation at the same time [64]. Thus, it is a typical type of RCP. We identify seven specific scenarios under this category. Among them, 39% of the respondents have participated in RCP to **work on the same local machine**. It is a very collaborative way of working that can make multiple developers use the same configuration to share and discuss information with each other more easily. Another common scenario is **front and back end collaboration**. We notice that 39% of the respondents have participated in this scenario, and 56% of the respondents believe that RCP is/may be helpful. This is because there are generally two types of tasks in the software development process, i.e., UI development and server-side development. These tasks need different skills and are usually undertaken by different roles, i.e., front-end developers and back-end developers. Connecting the front-end with the back-end is a challenge for many development teams [49]. RCP enables developers to browse others' code easily and is conducive to establishing communication so that it may mitigate this challenge. Moreover, 35% and 29% of the respondents admitted that they have participated in RCP for **prototyping** and **breaking down tasks**. As P2 said, *when we need to start a project, the team members, usually two or three people, work collaboratively in real-time to build the foundations, e.g., set up directories. Then, we can break down the tasks and further process them individually.* P12 also expressed similar opinions, *When we need to understand the project structure or assign tasks to everyone, we usually need to collaborate in real time.* We also find that some developers, especially students, participate in RCP for completing **school group projects**. Students majoring in computer science need to complete team-based collaboration projects to enhance learning and prepare for their future careers. The effect of RCP on this scenario has been widely recognized and agreed upon by 54% of the respondents. However, it should be noted that only one-third of developers with such experience believed RCP is helpful. It indicates that many of the users' needs are still not being met, and they face significant challenges when participating in RCP. In addition to the above scenarios, **coding competition** and **mob programming** are also possible scenarios for RCP. P3 and P6 specifically mentioned the concept of "mob programming". For example, P3 said, *people are looking for effective and pleasant ways to learn new programming languages. This format is mob programming. It turns out to be just that effective and pleasant.* Mob programming is a software development approach that involves the entire team working together on the same task simultaneously [52]. In this approach, team members take turns being the "driver," actively writing the code while the rest of the team provides input, suggestions, and real-time code reviews. The team collaboratively solves problems, makes decisions, and produces high-quality code. This approach is similar to pair programming but involves a larger group of developers.

(2) Group Debugging. Debugging is an essential process of programming and engineering. It allows developers to fix errors in a program before releasing it to the public [33]. However, it is one of the most challenging problems in software development [9]. Developers need a comprehensive understanding of the whole system and rich testing experience to locate the root cause of the errors. Since debugging requires considerable developers skills, collaborative debugging may be effective, especially for some tough cases. Therefore, **participating in group debugging** is also a scenario for RCP. We find that 64% of the respondents have this experience, but only 55% of the respondents think that RCP is or may be helpful for this scenario. As P6 said, *When there's a*

junior developer who runs into an error, I find it easier to use breakpoints to pinpoint the problem. I show them how to set up breakpoints and go through the points where it crashes. This way, I can help them understand what went wrong more clearly. This example also reflects the following scenario: interactive education.

(3) Interactive Education. This category contains five specific scenarios. Like other engineering courses, learning programming needs highly practical ability [22]. RCP can make the explanation more effective, particularly the "focus and follow" function of *Live Share*, which allows participants to follow up with their mentors easily. We find that the most common scenario is **quick assistance** (59% of the respondents have participated in this scenario), which usually happens during office hours. **Peer mentoring** is the second common scenario. As mentioned by P9, when mentors introduce mentees to a new codebase, feature area, technology, or any other aspect, they can utilize RCP to guide them through the project. This allows mentees to follow along with mentors but within their own personal IDE. RCP enables mentors to provide hands-on guidance and support, while mentees actively engage and learn within their familiar development environment. This approach not only enhances the learning experience but also promotes effective knowledge transfer and skill development. RCP can also serve as a way for **classroom lectures**, which may promote the birth of a new programming teaching mode. We find that 17% of the respondents have this experience, and 52% of the respondents think that RCP is or may be helpful for this scenario. In addition to the above scenarios, we also find two scenarios with fewer participants. The first one is **brown bags meeting**. It is effectively like peer mentoring but presented to an entire team and potentially more focused on socializing generally useful knowledge, as opposed to onboarding support and/or helping with a specific task. The second one is **developer streaming**. With the popularity of social media, more and more people want to share their life and knowledge through live broadcasts. Watching a programming live stream can help audiences quickly understand that even experienced programmers struggle, observe the little tips and tricks they use to speed up their programming or improve their code, and even communicate with streamers.

(4) Code Review. Code review traditionally starts only when developers submit their patches (or pull requests) and is conducted by reviewers asynchronously. Many times, the advice on the patches could be given earlier, not waiting for the complete patches to come. Therefore, there is potential value for teams to easily and continuously seek advice from their peers. With the emergence of RCP tools, reviewers can initiate informal reviews and technical discussions at any time to ensure the patches are in the right direction. This can potentially help subsequent patches complete quicker and help socialize knowledge across the team. For example, P11 said, *Before the code goes live, we often collaborate in real time for code review, usually about once a week on average. However, not all code review involves real-time collaboration. Typically, we only do real-time showcases and discussions within the IDE for critical algorithm reviews, and then figure out the best way to make any necessary changes.* However, regarding the preference for asynchronous code reviews in certain situations. Some developers (e.g., P9) find that coordinating time for real-time code reviews can be challenging, especially when there are no complex issues to discuss.

(5) Working Remotely. We find that 17% of the respondents choose this category as their most common scenario for conducting RCP. After all, **online job** is not a new work pattern for developers [4] and 48% of the respondents conducting RCP at home (see Figure 2). P8 as a software development intern, he said, *I am not currently employed in a full-time position, but I am undertaking an internship remotely, rather than in a permanent capacity.* Due to **COVID-19**, developers who initially worked in the office also started working from home. For example, P5 said, *I began using this (RCP tools) from 2020 when the coronavirus pandemic came in. During the COVID period, I worked with my teammates, but we could not meet with each other, so we collaborated remotely.* There are also 42% of the respondents who have participated in RCP in order to **run use case on other's**

machine. P5 indicates that *I need to be able to share my APIs with my teammates to let them know what I am working on and sometimes to run some commands or use cases as if we were using the same machine.*

(6) Technical Interview. When **conducting technical interviews**, RCP allows the interviewees to be in their own environment (including OS settings, such as accessibility) and would work equally as well for local or remote interviews. RCP can also trigger rich discussion content, such as step debugging and running tests, which cannot be realized by whiteboard discussions. We find that 51% of the respondents believe that RCP is or may be helpful in this scenario. For example, P7 said, *I work part-time as a software engineering interviewer, assessing candidates for positions at a company called ***. During these interviews, I utilize real-time collaboration tools like Replit or Code With Me, depending on the setup they have. These tools allow me to see the logic and thought process behind a candidate's coding as they write it.* P10 as an interviewee, he shared his experiences and said, *During interviews, the interviewer might give you questions and prompts in a web editor. It's crucial to stay engaged and think on your feet. By rephrasing your answers and sharing new perspectives, you can keep the conversation lively. Using real-time collaborative programming ensures that your thoughts are accurately captured. Good communication and interaction with the interviewer make the exchange more productive.*

Summary of Scenarios

We identify 18 scenarios in six categories for RCP. **Pair programming** is the most common and rich category, which contains seven scenarios, e.g., *prototyping*, *front and back end collaboration*, and *breaking down tasks*. **Interactive education** is another rich category, containing five scenarios, such as *quick assistance*, *peer mentoring*, and some emerging activities (e.g., *classroom lectures* and *developer streaming*). We also notice that **code review**, **group debugging**, and **technical interview** are other potential scenarios for RCP. Due to the prevalence of remote jobs and the impacts of COVID-19, developers may participate in RCP when they **work remotely**. On average, individual developers participated in six scenarios, but their attitudes toward the usefulness of the current RCP tools were significantly more negative than the expectations.

4.2 RQ2: Requirements

Our second research question investigates the developers' requirements when conducting RCP. We asked developers to select the three most important categories of requirements when participating in RCP. In total, 69 developers (67%) chose **Live Editing** as the most important, 30 chose **Sharing Terminals** as the second most important, and 22 chose **Focusing and Following** as the third. Table 6 shows the requirements of RCP, which contains 17 requirements belonging to seven categories. Most of the requirements are considered to be important but not supported well by current RCP tools, explaining the respondents' evaluation of usefulness in RQ1 to some extent. In the following, we explain each requirement but do not connect these requirements with different scenarios considering that these requirements are applicable to any scenario instead of a specific one.

(1) Live Editing. This is the most basic category of requirements of RCP. Once a guest has joined a collaboration session, the collaborators can immediately see each other's edits and selections in real-time, as shown in Figure 3. All participants can contribute, making it easy to iterate and rapidly nail down solutions. We obtain five requirements of **Live Editing**. The first one is **auto saving for interruptions**, which is agreed by 78% of the respondents. Network quality is critical for RCP,

Table 6. Requirements of RCP. A slash ("/") means that the requirement is derived from open-ended questions. All the Likert distributions have five columns, from left to right, indicating *strongly disagree*, *disagree*, *neutral*, *agree*, and *strongly agree*.

| Code ID	Category & Requirement	Likert Distribution	% Agreement	Conflict Factor
	Live Editing			
C19	Auto saving for interruptions		78	0.90
C20	Automatic Git	/	/	
C21	Permission separation of different participants		77	0.88
C22	Reminder of disconnect		77	0.76
C23	Tracking changes		87	0.89
	Sharing Terminals			
C24	Adaptation of terminal style on different machines		62	0.59
C25	Protection mechanism for visiting local files on terminal		73	0.77
C26	Sharing more information not only text		80	0.88
C27	User defined permissions for certain commands		61	0.53
	Focusing and Following			
C28	Focusing and following participants	/	/	/
	Sharing Servers			
C29	Sharing more server information to confirm a correct connection		68	0.80
	Group Debugging			
C30	User defined permission for debug button		63	0.68
	Chatting			
C31	Compatible with other chatting platforms		41	0.22
C32	Exporting the audio or chatting history		46	0.28
	Sharing More Artifacts			
C33	Screen demonstration	/	/	/
C34	Sharing the virtual machines and containers	/	/	/
C35	Sharing untracked but local files	/	/	/
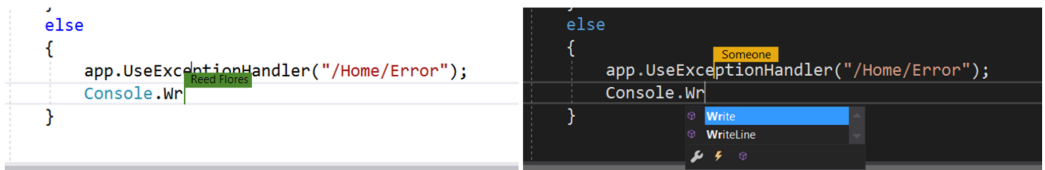
Fig. 3. Example of live editing in live share.

but it is always hard to ensure a stable network connection. Therefore, it is necessary to explicitly **remind participants of disconnection** and automatically save the modifications. **Permission separation of different participants** is also an essential requirement when conducting RCP. When multiple developers participate in software development synchronously, operations from different participants may cause conflicts or dependencies (such as deleting and opening the same

file simultaneously). Therefore, it is necessary to separate participants' permissions and define the priorities of different operations. P7 also mentioned that it is important to have different colors assigned to each person who is editing. Besides the above problem, it is hard to remember which developer conducted the modification. Thus, **tracking everyone's changes** is a requirement for RCP. With track changes, developers can view edits and comments from previously saved versions of code files to suggest further revisions and trace responsibilities. **Automatic Git** is its further requirement that we obtain through open-ended questions. Two respondents mentioned this requirement, as a respondent said, *it would be nice if the remote edits could still be attributed to the person who made them when committing code into git.*

(2) Sharing Terminals. In modern software development, developers frequently use command-line tools. Therefore, when developers conduct RCP, they may also need to share the terminal with other participants. This is the second most important category of requirements, which contains four requirements. The first requirement is **adaptation of terminal style on different machines**. The terminal display may be inconsistent due to different machine configurations. During the interview, P1 said, *when the host enters a long command line with a line break because the size of the guest terminal is unknown, the place of the line break may be inconsistent. For another example, sometimes there will be full-screen applications on the terminal, and this full-screen display will also have problems.* The current implementation of sharing terminals seems only transmit text information without format information, e.g., without showing color and highlight. This information is also necessary for participants to understand others' operations. Thus, the second requirement is **sharing more information, not only text**. The third requirement is **protection mechanism for visiting local files on terminal**. This requirement is mainly out of the protection of hosts. After all, sharing terminals may bring significant potential risks, e.g., deleting files by mistake or viewing unshared files. *Live Share* has noticed this issue, so it supports two ways to share a terminal, i.e., read-only and fully collaborative. Fully collaborative means collaborators can run commands and see the results as hosts. However, the interviewee said that *the current granularity of authority management is too coarse, and it is better to allow developers to customize the authority of different commands.* It reflects the last requirement—**user-defined permissions for certain commands**.

(3) Focusing and Following. Sometimes, developers may need to explain a problem or design that spans multiple files or locations in code. In these situations, following a colleague as they move throughout the project can be helpful. As P6 said, *It's a useful feature, especially when you want to teach or demonstrate something. Let's say you want to show someone a specific part of your code while they are working on another file. With this feature, you can easily guide them by asking them to follow along and they will be able to see exactly what you're pointing at. It's a great way to provide visual context and enhance the learning or collaborative experience.* P 11 also expressed a similar opinion, *The 'focusing and following' feature can highlight the areas being modified by the host, making it easier for participants to keep up..* Similarly, sometimes developers may want everyone in a collaboration session to come and look at something they are doing. Then, the host can ask that everyone focus their attention on her/him with a notification. Therefore, the relationship between "focusing" and "following" is active and passive. This is the third important requirement for RCP. It is inseparable from **Live Editing** and is necessary for interactive education.

(4) Sharing Servers. In addition to sharing terminals, developers might want to share local servers or services with other collaborators, e.g., databases. P12 said, *If the project involves a server, the 'sharing servers' feature is still quite important. It helps facilitate collaboration and keeps everyone on the same page when working with server-related tasks.* Regarding this category, developers propose a requirement—**sharing more server information to confirm a correct connection**. For example, P5 said, *Live Share may change the port for security reasons or because the port is already*

occupied. It may be a little bit confusing. Therefore, when confirming the connection, it would be better if we could share more information with our partners, not just the port number. P7 also expressed similar requirements—*Previously, we used a tool called JetBrains Space, which had its servers and used separate APIs to authenticate users individually. However, during collaboration, there were times when someone would get kicked off due to a network change. This caused confusion because each time a developer rejoined the collaboration, the authentication point would change. To address this issue, it would be fantastic to have a shared key or link that provides more information between collaborators. This way, we can avoid the confusion caused by changing authentication points and ensure a smoother collaboration experience.* More than half of the respondents agree with this requirement, indicating the deficiencies of the current port-sharing mechanism.

(5) Group Debugging. When conducting RCP, developers may collaboratively troubleshoot problems involving various activities. Each collaborator can investigate different variables, jump to different files in the call stack, inspect variables, and even add or remove breakpoints. When developers become addicted to their own debugging process, they may overlook the actions of other collaborators and even bring conflicts. Moreover, debugging privilege is a security policy setting that allows developers to attach a debugger to a process or to the kernel, which is especially important for sensitive and critical software, such as operating system components. Therefore, regarding this requirement, developers hope to **customize the permission for debug button**, e.g., permission management of the breakpoint setting.

(6) Chatting. Effective RCP heavily relies on communication among collaborators. As P8 said, *Because there is a situation where someone is writing code and another person may not understand it, it is important to find a solution to ensure effective communication. In this case, utilizing chat and voice communication can be highly beneficial.*. However, given the abundance of communication tools already utilized by developers in their daily lives, the necessity of communication tools specifically designed for RCP becomes uncertain. As mentioned by P6, *I think this function is useless. We are already using platforms like Skype, Slack, and Google Meet. I think they should completely remove this function because no one uses it.*. However, P11 expressed the opposite opinion, *I think the chatting feature is still useful. Even though we have many chatting apps already, the chatting feature in Live Share helps me separate work from personal life.* P9 made an intriguing observation regarding the importance of text communication during the RCP process. While voice and video communication may not be essential, text communication holds significance due to the need for code snippet sharing. P9 pointed out *although developers can share code snippets within the coding editor, they often hesitate to do so due to the additional step of deleting the shared code afterward.* During interviews, we identify two requirements from developers who expressed the usefulness of chat functionality within the RCP tools. However, the respondents expressed a "neutral" viewpoint on these requirements in the online survey, indicating divergent opinions. This suggests that these requirements are not immediately urgent or critical. The first one is **compatible with other chatting platforms** because developers may be more accustomed to using existing chat tools. A developer in the interview said, *they often have established communication through phone, WeChat, and other chat tools before conducting RCP, so it is natural for them to continue using the previous chat tools during RCP.* The second one is **exporting the audio or chatting history** to ensure the discussion content cannot be lost when the RCP process is finished. Respondents have the greatest divergence regarding this requirement, which means further investigations on its necessity are needed.

(7) Sharing More Artifacts. Developers also hope to share more artifacts, not only code files, terminals, and servers. Through the online survey, we notice that developers also hope to share **screen demonstration**, **virtual machines and containers**, and **untracked but local files**. As for the last requirement, the developer explained that *I hope to open the browsing permission for*

some files, which are not in the tracked project but are referenced by the project. Or users can define which files can be accessed in addition to the project itself. We often need to browse some file headers, which are not visible in Live Share. These requirements point out the directions for the future improvements of RCP tools.

Summary of Requirements

We identify 17 requirements of RCP belonging to seven categories, most of which are agreed upon by the respondents. The most important category of requirements is **live editing**. Developers propose five requirements under this category, e.g., *auto saving for interruptions*, *tracking changes*, and *automatic Git*. Because the command-line tools are frequently used during software development, developers also hope to **share terminals**. Most of its requirements are related to displaying and privacy. In addition to the above requirements, **focusing and following**, **sharing servers**, **group debugging**, and **chatting** are also the categories of requirements of RCP. However, developers' opinions are divergent on certain requirements, e.g., *exporting the audio or chatting history*. We notice that developers are not satisfied with the sharing content supported by the existing RCP tools, and they expect to **share more artifacts**, such as *virtual machines and containers*.

4.3   RQ3: Challenges

This question aims to reveal developers' challenges when participating in RCP. We identify 19 challenges belonging to seven categories, as shown in Table 7. To understand the seriousness of these challenges, we ask the developers to choose the three most serious challenges they face and evaluate each challenge according to their experiences. We find that the top three most serious categories of challenges are **Lagging**, **Permission and Conflict**, and **Software Defect**. Some challenges are consistent with or closely related to the requirements identified in RQ2, e.g., C43: *cannot resize the terminal* and C24: *adaptation of terminal style on different machines*. However, the codes in RQ3 focus more on the difficulties and terrible experiences encountered by developers instead of their demands. We discuss each of the challenges in detail.

  (1) Lagging. We observed that lagging poses the most significant category of challenges in RCP. When multiple programmers are simultaneously working on the same codebase, any delay in the responsiveness of the coding environment can impede productivity and collaboration. Lagging can take different forms during collaborative programming, and we have identified three challenges associated with lagging: **slow connection**, **unstable network**, and **different debugging speed on different PCs**. Among these, **unstable network** stands out as the most serious challenge, as agreed upon by 71% of the respondents. This is because RCP tools demand higher network requirements compared with other real-time collaboration editors. The information transmitted is more extensive, including rich content beyond plain text, and errors can have more severe consequences, such as program crashes. However, ensuring that all collaborators consistently have high-quality networks, such as high speeds and ample bandwidth, is a difficult task. Moreover, it is worth noting that lagging is not solely caused by a poor network connection. As expressed by P4, *even if we use a P2P connection during our collaboration, of course, you can choose to use their server as a proxy connection, but the terminal experience is still very slow*. P7 also expressed similar opinions, *Sometimes we've done speed tests to make sure the network is working fine, and the bandwidth seems to be good. But there are times when I'm using Live Share for collaboration while also being on a video call, and that's when the problem occurs. It seems like the tool itself has a problem with managing resources efficiently and ends up competing for them.*

Table 7. Challenges of RCP. A slash ("/") means that the challenge is derived from open-ended questions. All the Likert distributions have five columns, from left to right, indicating *strongly disagree*, *disagree*, *neutral*, *agree*, and *strongly agree*

Code ID	Category & Challenge	Likert Distribution	% Agreement	Conflict Factor
	Lagging			
C36	Different debugging speed on different PCs		44	0.30
C37	Slow connection		63	0.59
C38	Unstable network		71	0.66
	Permissions and Conflicts			
C39	Conflicting when editing the same place		57	0.38
C40	Unable to see untracked but related file		54	0.44
C41	Undoing other person's changes using Ctrl Z		61	0.50
	Software Defects			
C42	Audio not working	/	/	/
C43	Cannot resize the terminal		45	0.31
C44	Fail to share the correct server		48	0.30
C45	Fail to start chat		30	0.12
C46	Follow function failure		40	0.10
C47	Losing changes after interrupts		66	0.52
	Inconsistencies in Thinking Habits and Coding Styles			
C48	Different coding styles and ways of solution		50	0.36
	Adaptation Problems			
C49	Incompatible with other tools/plugins		65	0.54
C50	Cannot adapt to the configuration of other PCs		52	0.45
C51	Inconsistent display on different machines		53	0.44
C52	MAC vs WINDOWS command key	/	/	/
	Safety and Privacy			
C53	Safety and privacy of sharing terminals		60	0.51
	Unclear about Participant Status			
C54	Unknown of offline information		57	0.49

(2) Permissions and Conflicts. Because multiple people are working on the same project simultaneously, managing the permissions of different collaborators and avoiding conflicts are the challenges of RCP. We identify three challenges that are agreed upon by the respondents. First, **conflicts may occur when multiple people edit the same place**. This is a common challenge of real-time collaboration [44]. Several classical real-time collaboration algorithms have been proposed by researchers in order to transmit changes between users with the goal of "eventual consistency", e.g., **Operational Transformation (OT)** [58] and **Conflict-free Replicated Data Type (CDRT)** [50]. Although these algorithms are relatively mature, there are still challenges in practical application. Second, developers complain that they **cannot see untracked but related files**. During the process of software development, developers not only need to view the internal project files but also need to browse external files. This is a common scenario in software development. For example, developers may need to check the information on import packages and API documentation. However, the current RCP tools only support sharing the tracked files in a project. Therefore, the related files outside the projects cannot be shared or viewed with collaborators. This challenge

may reduce the efficiency of the collaboration. The third challenge is **undoing other person's changes using Ctrl Z**. It is essentially a permission-related issue, agreed by 62% of the respondents. *Live Share* currently supports developers to undo the last change of the system instead of their own changes through Crtl Z, which is different from the habits of developers. Therefore, developers may be confused about this operation.

(3) Software Defects. RCP is an emerging demand in software development. Although multiple RCP tools have been released, these tools are not stable. Take *Live Share* as an example. Up to November 2023, developers have reported 4,786 issues about *Live Share*, and 114 of them are still unsolved.[10]. We identify six challenges, e.g., **audio not working**, **cannot resize the terminal**, and **fail to start chat**. These issues are related to developers' requirements (i.e., what we identified in RQ2) and thus can directly affect the user experience of participating in RCP.

(4) Inconsistencies in Thinking Habits and Coding Styles. This category is the only challenge that originates from the collaboration mechanism instead of RCP tools. Respondents are neutral about this challenge, and their opinions are divergent. This is because this challenge is closely related to collaborators' technical skills and collaboration experiences. Thus, collaborators may be happy in cooperation, or they may experience bad experiences. The basic idea behind RCP is that multiple minds are better than one. Through multiple developers collaborating in real time, they can catch each other's mistakes and come up with better code solutions faster. However, this is not always the case in reality because it can be hard to find multiple developers who share the same vision. For example, multiple developers with different ideas about how things should be done may work together on software development in real time. This usually leads to friction about best coding practices that can easily escalate into big arguments over style preferences or coding conventions [30]. P10 especially expressed this kind of worry and he said, *When several people code together, it can get pretty confusing. Unlike writing documents, changing the code can affect the whole program, not just a small part. If you rename a variable, it might cause unexpected problems in the entire program. Plus, everyone has their own way of solving problems, which makes things even more complicated. It's definitely a big challenge.*

(5) Adaptation problems. Developers who participate in RCP may use different local machines. These machines usually have different configurations, e.g., different operating systems and screen resolutions. Thus, when one developer's operation is transmitted to different developers' screens, inconsistency may occur due to adaptation problems. This category has three challenges, which are all agreed upon by the respondents. A developer in the open-ended question also mentioned, "**the command keys of MAC and WINDOWS are not consistent**". In addition to the adaptation problems of different local machines, as a plugin, *Live Share* needs to **compatibly work with other tools/plugins** in the *VS Code* ecosystem to achieve rich functions. However, it is not satisfactory at present. P1 complained that *the debug function of VS Code is usually provided by plugins, rather than the function of VS Code itself. Although the host has clicked the debug button, the guests cannot witness the operation process since the plugin's compatibility with Live Share is not that excellent. Sometimes the debug function is not activated when someone clicks the button to begin debugging. Sometimes the speed of the debug is different on different collaborators' machines. There are often such strange problems.*

(6) Safety and Privacy. The safety and privacy of information in collaborative development are crucial to ensure a safer environment for RCP. This challenge is mainly related to the **function of sharing terminals**. The command-line interface is a powerful and handy utility for administering an operating system. It provides a fast and versatile way of running the system. Although it helps manage systems, the command line is fraught with risks. Running wrong commands can cause

---

[10]https://github.com/MicrosoftDocs/live-share/issues

harm and irrecoverable damage to the system. For example, delete important files by mistake using *rm -rf*, which makes it nearly impossible to recover these files. Therefore, sharing terminals with collaborators, especially sharing full access permissions, is a high-risk operation. In addition to safety problems, it may also bring privacy problems. For example, collaborators can browse any folders and files on the host's computer through the command line. Therefore, sharing terminals with others needs developers' great caution.

(7) Unclear about Participant Status. Keeping abreast of the status of the collaborators is necessary to ensure the smooth execution of RCP, especially when there are multiple collaborators. However, 57% of the respondents complained that they are **unknown of the offline information of their partners**. Live Share's information prompt for offline status only can be explicitly perceived by guests when the host loses network connection. For example, if guests attempt to use Live Share when the host is offline, they will be unable to join the session until the host is online again. Additionally, once collaboration stops (e.g., the host closes the editor, goes offline, or stops sharing), further actions or file access by the guests are immediately disabled. However, the host cannot receive an obvious notification when a guest loses network connection. When the host is immersed in development tasks, this issue becomes severe because it is more challenging to know the offline status of the guests timely. This challenge seriously affects the efficiency of RCP and reduces the collaborative experience. It indicates that the RCP tool needs to update the status of collaborators more timely and explicitly.

Summary of Challenges

We identify 19 challenges belonging to seven categories. The most serious category of challenge is **lagging**, which directly affects whether RCP can be successfully conducted. The second serious category is **permissions and conflicts**. It involves three challenges, e.g., *conflicting when editing the same place* and *undoing other person's changes using Ctrl Z*. The third serious category is **software defects**. We identify six defects, which means that the current RCP tools can be improved greatly. We also identify other categories of challenges, i.e., **inconsistencies in thinking habits and coding styles**, **adaptation problems**, **safety and privacy**, and **unclear about participant status**. These challenges bring rich insights into the ways to improve current RCP tools.

---

5 DISCUSSION AND IMPLICATION

This study investigates how developers participate in RCP, providing valuable insights into the scenarios, requirements, and challenges developers face in this collaborative mode. These findings provide a comprehensive understanding of the RCP mechanism and have implications for various stakeholders involved in software development.

5.1 Discussion

Substantial studies focus on collaborative software development [1, 12, 25, 40]. However, most of these studies investigate asynchronous collaborative programming. For example, researchers try to reveal the requirements [11, 14, 34] and challenges [31, 65] faced by developers in distributed development, and develop tools to optimize the collaboration process [19, 25, 61]. The few existing studies on RCP investigate the benefits of RCP [10, 16, 63] or design tools to address specific RCP challenges in certain scenarios [23, 32], lacking a comprehensive understanding of how developers participate in RCP.

Our study bridges this knowledge gap by examining the scenarios, requirements, and challenges of developer participation in RCP. Although RCP can alleviate particular challenges present in traditional asynchronous collaborative programming [15, 45], such as enhancing the richer information transfer and faster feedback, it is important to note that some challenges still persist in RCP. One such challenge is team awareness [24, 57] ("unclear about participant status" in our study), which can be significantly impacted by network quality in RCP. Poor network connectivity can hinder effective communication and impede team members' ability to stay informed about each other's actions and progress. "Inconsistencies in thinking habits and coding style" is also a critical challenge in collaborative software development, whether it is real-time or non-RCP.

Although we set three research questions to separately investigate the scenarios, requirements, and challenges, these three dimensions are interconnected and mutually influence each other in the context of RCP. Firstly, understanding the scenarios in which developers engage in RCP is crucial as it lays the foundation for identifying the specific requirements and challenges. Different scenarios, such as "pair programming", "group debugging", and "interactive education", present unique contexts and objectives for collaboration. The requirements and challenges faced by developers in each scenario are shaped by the nature and goals of the collaborative activities. Secondly, while the requirements are generally applicable to various scenarios, they are influenced by the scenarios in which developers collaborate. For example, in "pair programming", the requirement for "effective live editing" and "seamless chatting" between developers is critical. Similarly, in "group debugging", the ability to "share debugging information and synchronize actions" becomes an important requirement. The requirements are driven by the specific needs and goals of the collaborative activities in each scenario. Lastly, the challenges encountered by developers in RCP are closely tied to both the scenarios and the requirements. For instance, challenges related to "network connectivity and latency" may be riskier for scenarios that have a large amount of data transmitted and high requirements for latency, e.g., "classroom lectures". Similarly, challenges associated with coordinating efforts and resolving conflicts (e.g., "inconsistencies in thinking habits and coding style") may arise when developers conduct "pair programming". The requirements imposed by the scenarios often shape these challenges, and addressing them is essential to ensure effective collaboration.

In summary, our study contributes to the research on distributed collaborative development by detailed investigation of the RCP mechanism. We offer valuable insights into how developers engage in RCP, paving the way for advancements in techniques, tools, and practices to enhance productivity and effectiveness.

5.2 Implication

Our study has rich implications for practice, tool design, and research. During the survey and interview, many developers expressed their interest in our study, e.g., *the survey allowed me to express my requirements for RCP well, especially challenges I faced, it's a useful and interesting survey. I hope it would help improve something*, and *the survey is a good way of feedback and had almost all the things included which are used, faced or which people want in their collaborative coding environment.*

5.2.1 Practical Implication. The direct beneficiary of our study is developers who participate in RCP. Through analysis, we carefully investigate RCP regarding scenarios, requirements, and challenges. Our findings can promote the smooth and efficient progress of RCP. In particular, we sort out various scenarios of RCP (RQ1), including six scenarios and 18 scenarios. For such a neo-type software development mode, the clarification of its scenarios is necessary. It can help developers realize that they can use RCP tools to assist the collaboration process in these scenarios and even promote the emergence of new modes of collaboration. For example, traditionally, pair

programming needs two developers to work on the same code together at the same place [26]. However, most office spaces are designed for independent work rather than collaboration. There are few shared workspaces or meeting rooms, and often there is not enough space for two chairs next to each other at desks set up for individual work. Our results reveal that pair programming is a typical scenario for RCP. Therefore, by utilizing RCP tools, developers can conduct pair programming in different locations instead of sitting together. Take another example, COVID-19 has promoted the development of online education, but traditional live/recorded online education lacks timely and effective communication and interaction. Thus, traditional online education is unsuitable for courses with highly practical requirements, such as software development. We find that RCP is suitable for interactive education, which provides insights for promoting the change of traditional education way.

The findings of RQ2 and RQ3 highlight a set of requirements and challenges for RCP, which can help to understand the core requirements of developers and the problems that should be paid special attention to during the RCP process. For example, we find that although "sharing terminals" is the second essential requirement for RCP, it is a high-risk operation. It may bring safety and privacy issues. Therefore, developers need to use this function with caution and carefully evaluate whether the collaborators can be trusted, especially sharing full access permission. We also notice that lagging is the most severe challenge for RCP. To mitigate this problem and avoid losing changes after an interruption, developers need to ensure a good network environment as much as possible, e.g., avoid applications requiring large bandwidth running simultaneously. At the same time, maintain positive communication with collaborators to understand whether the state of the host is consistent with what the guests see. Our findings also provide rich insights for optimizing RCP tools, which can indirectly improve the developer's experience of participating in RCP.

5.2.2 Design Implication. Although various tools have emerged to support RCP, we observe that the existing users' attitudes toward the usefulness of the current RCP tools were significantly more negative than the expectations of potential users (refer to RQ1). Therefore, there is still much room for improvement. The root cause is due to little knowledge of this collaboration mechanism and the unclear support situation of current RCP tools. Our findings preliminarily reveal this mechanism and users' experience of current RCP tools, which can bring rich implications for RCP tool designers.

First, RQ1 reveals various scenarios of RCP and their frequency and usefulness. Based on our findings, designers can thoroughly excavate the pain points of users in specific scenarios. Then, they can focus on some typical scenarios and develop specific RCP tools to meet different users' personalized needs better. For example, for the scenario of "technical interview", designers can add the *video recording* function and "*anti-cheating* mechanism. For "interactive education", because this scenario usually involves a large number of participants, designers should first consider how to ensure that the tool can still run normally when the traffic is high. Then, designers can add some specific functions in combination with the characteristics of classroom teaching, such as the *raise hand* function and *whiteboard* function. For "developer streaming", designers can add more *social media elements* such as follow, like, comment, and donation and add *hyperlinks to developers' social media accounts*, like Twitter, GitHub, and LinkedIn.

Second, RQ2 reveals a set of requirements for RCP. On the one hand, the importance of these requirements can provide a reference for designers in determining task priorities because we find that there are still a large number of issues unresolved in the *Live Share* community (see Section 4.3). We find that "live editing" is the most crucial requirement of RCP. Therefore, designers should give priority to the issues related to this function. On the other hand, we identified 17 requirements, most of which are not supported well by current RCP tools. These requirements can

provide detailed references for optimizing RCP tools, e.g., *sharing untracked but local files*, *screen demonstration*, *user-defined permission for debug button*, and *automatic Git*.

Third, RQ3 identifies 19 challenges of RCP belonging to seven categories. The most severe challenge reported by participants is "lagging". This implies that the success and user experience of RCP heavily depend on underlying internet infrastructure and resource scheduling. This finding has two important implications. Firstly, it emphasizes the significance of network reliability as a crucial aspect when designing, implementing, and using RCP tools. Despite advanced features and functionalities, these tools' effectiveness can be significantly impacted by factors beyond their control, such as network disruptions or high latency. Hence, developers and tool designers must be mindful of the limitations imposed by network connectivity and strive to optimize resource scheduling accordingly (e.g., optimal utilization of computing resources, load balancing, scalability, and priority management). Secondly, this finding highlights the necessity for internet infrastructure improvements to facilitate seamless RCP experiences. Unstable network connections can cause synchronization problems, delayed updates, and hindered communication between developers. Advocating for enhanced network infrastructure, including increased bandwidth, reduced latency, and improved stability, becomes crucial in facilitating smoother and more efficient real-time collaboration.

Our findings also indicate that current tools may need to be more mature to support RCP. Many functions need to be optimized. For example, *better adapt to other tools and plugins*, *ensure consistent display on different machines*, and *explicitly remind offline status*. Based on the challenges we have discovered, designers can conduct more in-depth interviews with users so as to formulate a reasonable improvement plan.

5.2.3   Research Implication. Our study provides a starting point for guiding developers to conduct RCP and providing ways to optimize RCP tools. More work is necessary on specific research topics. For example, explore more detailed requirements and challenges under different scenarios and investigate why respondents' evaluation of the usefulness of RCP in different scenarios differs significantly from potential users' expectations. It is also necessary to investigate more developers with RCP experiences and invite developers who have used other RCP tools to complement and refine our findings. Future studies can also utilize our findings to optimize current RCP tools and even build new social and collaborative systems/tools to facilitate RCP. For example, one possible avenue for future research would be to investigate ways to minimize the impact of unreliable network connections in RCP. This could involve exploring alternative communication protocols that can withstand network disruptions or creating adaptive RCP tools that can adapt to changing network conditions. Furthermore, collaboration among tool developers, network providers, and researchers can result in progress in both RCP tools and internet infrastructure, ultimately enhancing the collaborative programming experience for developers across the globe.

6   THREATS TO VALIDITY

6.1   Threats to Internal Validity

Internal validity concerns the threats to how we perform our study [39]. The first threat is related to the **subjectivity of the data classifications**. To avoid this threat, we use open card sorting in which all analysis is thoroughly grounded in the data collected. Additionally, we exhaustively discuss the analysis and results to reach an agreement.

The second threat is related to the **sample bias of the participants** in our study. Self-selection bias is a typical threat of surveys and interviews introduced into a research project when participants choose whether or not to participate in the project [27]. To counteract this effect, we sent the interview and survey invitations to different developers with RCP experiences to ensure the

participants' diversity. By analyzing the demographic information of these participants, we can confirm that they indeed have various backgrounds (see Table 3 and Figure 2).

The third threat is related to **data representativeness**. We probably do not uncover all potential scenarios, requirements, or challenges of RCP or provide comprehensive explanations, despite carefully reading the official documents of *Live Share*, attempting the most popular RCP tools, and conducting interviews and an online survey with practitioners from a variety of backgrounds. To mitigate this risk, we employed various strategies to increase our response rate, such as sending personalized invitations, maintaining participant anonymity, emphasizing the importance of their feedback, and implementing an incentive mechanism. We also included open-ended questions in the questionnaire to allow developers to freely express their opinions. While we did obtain approval from the ethics board for the study, we understand the potential risks of reaching out to Live Share users via e-mail, especially if their e-mail addresses are intended solely for communication with fellow developers. Given the absence of a direct means to contact *Live Share* users, we felt that e-mail outreach was the most viable option. To mitigate this risk, we only sent out questionnaires once. Additionally, through the authors' own social networks, we involved 37 participants in our questionnaire survey, thereby expanding the number of survey responses. In the future, researchers could consider leveraging community resources, such as specific crowdsourcing platforms, or targeting a small group of initial respondents and then expanding the respondent pool through their social connections. This approach can enable future researchers to investigate more developers with RCP experiences, thus complementing and refining our findings.

6.2 Threats to External Validity

Since we mainly focus on the developers who have used *Live Share*, some findings may not generalize to the developers who have used other tools to conduct RCP. Although we do not anticipate significant differences in these RCP tools, additional research is necessary to investigate the transferability of the results. In fact, the demographic information of our survey indicates that some of the respondents also have used other RCP tools, e.g., *Code with Me* and *Replit* (see Figure 2). This can mitigate this risk to a certain degree. Besides, since *Live Share* is a preeminent RCP tool developed by the world's top IT enterprise—Microsoft, most of our findings (e.g., scenarios and requirements) have great reference value and guidance significance to understanding RCP mechanism and optimizing RCP tools.

7 CONCLUSIONS

This article takes the first steps toward understanding the RCP mechanism. By combining interviews and an online survey with *Live Share* users, we identify the scenarios, requirements, and challenges of RCP, as well as developers' opinions on these findings, which can deepen our understanding of RCP and point out the directions for future optimizations of relevant tools. We find that developers participate in RCP in six categories of scenarios (18 scenarios). The most common category is *pair programming*, followed by *group debugging*, *interactive education*, *code review*, and so on. We identify 17 requirements of RCP belonging to seven categories. The top three most essential categories of requirements are *live editing*, *sharing terminals*, and *focusing and following*. Most of the requirements (e.g., *automatic Git*) are not well supported by the current tools, indicating much room for improvement. As for challenges, we obtain 19 challenges belonging to seven categories, e.g., *lagging* and *permissions and conflicts*. These challenges and their seriousness can help formulate optimization schemes for RCP tools to improve the user experience in the RCP process. Because various RCP tools are constantly emerging, investigating the RCP mechanism is urgent and significant. Our findings preliminarily reveal the current status of this neotype collaborative development mode, which has rich implications for practitioners, tool designers, and researchers.

ACKNOWLEDGEMENTS

REFERENCES

[1] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. 2018. Understanding semi-structured merge conflict characteristics in open-source java projects. *Empirical Software Engineering* 23, 4 (2018), 2051–2085.

[2] Flora Amiti. 2020. Synchronous and asynchronous e-learning. *European Journal of Open Education and E-Learning Studies* 5, 2 (2020), 60–70.

[3] Tim Anderson. 2021. A swarm in May is worth a load of hay, is it? JetBrains Code With Me collaborative programming tool released. (2021). Retrieved Dec, 2022 from https://www.theregister.com/2021/04/07/jetbrains_code_with_me/

[4] Saulius Astromskis, Gabriele Bavota, Andrea Janes, Barbara Russo, and Massimiliano Di Penta. 2017. Patterns of developers behaviour: A 1000-hour industrial study. *Journal of Systems and Software* 132 (2017), 85–97.

[5] Nicholas Bello. 2022. Replit. (2022). Retrieved Dec, 2022 from https://research.contrary.com/reports/replit

[6] Pernille Bjorn, Jakob Bardram, Gabriela Avram, Liam Bannon, Alexander Boden, David Redmiles, Cleidson de Souza, and Volker Wulf. 2014. Global software development in a CSCW perspective. In *Proceedings of the Companion Publication of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*. 301–304.

[7] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. 2014. How do centralized and distributed version control systems impact software changes?. In *Proceedings of the 36th International Conference on Software Engineering*. 322–333.

[8] Ray L. Chambers and Chris J. Skinner. 2003. *Analysis of Survey Data*. John Wiley & Sons.

[9] An Ran Chen. 2019. An empirical study on leveraging logs for debugging production failures. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. IEEE, 126–128.

[10] Charles H Chen and Philip J Guo. 2019. Improv: Teaching programming at scale via live coding. In *Proceedings of the Sixth (2019) ACM Conference on Learning@ Scale*. 1–10.

[11] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S Lasecki, and Steve Oney. 2017. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 6220–6231.

[12] Kattiana Constantino, Shurui Zhou, Mauricio Souza, Eduardo Figueiredo, and Christian Kästner. 2020. Understanding collaborative software development: An interview study. In *Proceedings of the 15th International Conference on Global Software Engineering*. 55–65.

[13] Bernardo José da Silva Estácio and Rafael Prikladnicki. 2015. Distributed pair programming: A systematic literature review. *Information and Software Technology* 63 (2015), 1–10. DOI:https://doi.org/10.1016/j.infsof.2015.02.011

[14] Daniela Damian. 2001. An empirical study of requirements engineering in distributed software projects: Is distance negotiation more effective?. In *Proceedings Eighth Asia-Pacific Software Engineering Conference*. IEEE, 149–152.

[15] Mariam El Mezouar, Daniel Alencar da Costa, Daniel M. German, and Ying Zou. 2021. Exploring the use of chatrooms by developers: An empirical study on slack and gitter. *IEEE Transactions on Software Engineering* 48, 10 (2021), 3988–4001.

[16] Hongfei Fan, Kun Li, Xiangzhen Li, Tianyou Song, Wenzhe Zhang, Yang Shi, and Bowen Du. 2019. CoVSCode: A novel real-time collaborative programming environment for lightweight IDE. *Applied Sciences* 9, 21 (2019), 4642.

[17] Hongfei Fan and Chengzheng Sun. 2012. Achieving integrated consistency maintenance and awareness in real-time collaborative programming environments: The CoEclipse approach. In *Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, 94–101.

[18] Hongfei Fan, Chengzheng Sun, and Haifeng Shen. 2012. ATCoPE: Any-time collaborative programming environment for seamless integration of real-time and non-real-time teamwork in software development. In *Proceedings of the 2012 ACM International Conference on Supporting Group Work*. 107–116.

[19] Jon Froehlich and Paul Dourish. 2004. Unifying artifacts and activities in a visual tool for distributed software development teams. In *Proceedings of the 26th International Conference on Software Engineering*. IEEE, 387–396.

[20] Erik Frøkjær, Morten Hertzum, and Kasper Hornbæk. 2000. Measuring usability: Are effectiveness, efficiency, and satisfaction really correlated?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 345–352.

[21] D Randy Garrison, Martha Cleveland-Innes, Marguerite Koole, and James Kappelman. 2006. Revisiting methodological issues in transcript analysis: Negotiated coding and reliability. *The Internet and Higher Education* 9, 1 (2006), 1–8.

[22] Carlo Ghezzi and Dino Mandrioli. 2005. The challenges of software engineering education. In *International Conference on Software Engineering*. Springer, 115–127.

[23] Max Goldman, Greg Little, and Robert C. Miller. 2011. Real-time collaborative coding in a web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. 155–164.

[24] Carl Gutwin, Reagan Penner, and Kevin Schneider. 2004. Group awareness in distributed software development. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. 72–81.

[25] Anja Guzzi, Alberto Bacchelli, Yann Riche, and Arie Van Deursen. 2015. Supporting developers' coordination in the IDE. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. 518–532.

[26] Jo E. Hannay, Tore Dybå, Erik Arisholm, and Dag IK Sjøberg. 2009. The effectiveness of pair programming: A meta-analysis. *Information and Software Technology* 51, 7 (2009), 1110–1122.

[27] James J. Heckman. 1990. Selection bias and self-selection. In *Econometrics*. Springer, 201–224.

[28] James D. Herbsleb, Audris Mockus, Thomas A. Finholt, and Rebecca E. Grinter. 2000. Distance, dependencies, and delay in a global collaboration. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*. 319–328.

[29] Onur Hisarciklilar and Jean-François Boujut. 2006. Reducing the" information gap" between synchronous and asynchronous co-operative design phases. In *Virtual Concept'06*. 67.

[30] Branimir Hrzenjak. 2022. 5 common challenges of pair programming. (May 2022). Retrieved Dec. 26, 2022 from https://www.shakebugs.com/blog/challenges-of-pair-programming/

[31] Miguel Jiménez, Mario Piattini, and Aurora Vizcaíno. 2009. Challenges and improvements in distributed software development: A systematic review. *Advances in Software Engineering* 2009 (2009). 1–14.

[32] Janne Lautamäki, Antti Nieminen, Johannes Koskinen, Timo Aho, Tommi Mikkonen, and Marc Englund. 2012. CoRED: Browser-based collaborative real-time editor for java web applications. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*. 1307–1316.

[33] Joseph Lawrance, Christopher Bogart, Margaret Burnett, Rachel Bellamy, Kyle Rector, and Scott D Fleming. 2010. How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering* 39, 2 (2010), 197–215.

[34] Leandro Lopes, Rafael Prikladnicki, Jorge Audy, and Azriel Majdenbaum. 2005. Requirements specification in distributed software development a process proposal. *Requirements Specification in Distributed Software Development a Process Proposal. Porto Alegre, RS, Brazil* (2005).

[35] Yifan Ma, Batu Qi, Wenhua Xu, Mingjie Wang, Bowen Du, and Hongfei Fan. 2023. Integrating real-time and non-real-time collaborative programming: Workflow, techniques, and prototypes. *Proceedings of the ACM on Human−Computer Interaction* 7, GROUP (2023), 1–19.

[36] Yifan Ma, Zichao Yang, Brian Chiu, Yiteng Zhang, Jinfeng Jiang, Bowen Du, and Hongfei Fan. 2021. Supporting cross-platform real-time collaborative programming: Architecture, techniques, and prototype system. In *Collaborative Computing: Networking, Applications and Worksharing: 17th EAI International Conference, CollaborateCom 2021, Virtual Event, October 16-18, 2021, Proceedings, Part II 17*. Springer, 124–143.

[37] Jetbrains Marketplace. 2023. Code With Me. (14 Nov. 2023). Retrieved Nov, 2023 from https://plugins.jetbrains.com/plugin/14896-code-with-me

[38] Visual Studio Marketplace. 2023. Microsoft Visual Studio Live Share. (14 Nov. 2023). Retrieved Nov, 2023 from https://marketplace.visualstudio.com/items?itemName=MS-vsliveshare.vsliveshare

[39] Joseph Maxwell. 1992. Understanding and validity in qualitative research. *Harvard Educational Review* 62, 3 (1992), 279–301.

[40] Shane McKee, Nicholas Nelson, Anita Sarma, and Danny Dig. 2017. Software practitioner perspectives on merge conflicts and resolutions. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 467–478.

[41] Patrick E. McKnight and Julius Najab. 2010. *Mann−Whitney U Test*. John Wiley & Sons, Ltd, 1–1. https://doi.org/10.1002/9780470479216.corpsy0524   DOI : arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470479216.corpsy0524

[42] Audris Mockus, Roy T Fielding, and James Herbsleb. 2000. A case study of open source software development: The apache server. In *Proceedings of the 22nd International Conference on Software Engineering*. 263–272.

[43] Kathryn E. Newcomer, Harry P. Hatry, and Joseph S. Wholey. 2015. Conducting semi-structured interviews. *Handbook of Practical Program Evaluation*. 492–505.

[44] Antti Nieminen. 2012. Real-time collaborative resolving of merge conflicts. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*. IEEE, 540–543.

[45] Esteban Parra, Mohammad Alahmadi, Ashley Ellis, and Sonia Haiduc. 2022. A comparative study and analysis of developer communications on slack and Gitter. *Empirical Software Engineering* 27, 2 (2022), 1–33.

[46] Fabio Petrillo, André Suslik Spritzer, Carla Maria Dal Sasso Freitas, and Marcelo Soares Pimenta. 2011. Interactive analysis of likert scale data using a multichart visualization tool. *IHC+ CLIHC* 67 (2011), 358–368.

[47] Julie Ponto. 2015. Understanding and evaluating survey research. *Journal of the Advanced Practitioner in Oncology* 6, 2 (2015), 168.

[48] Paul Ralph, Sebastian Baltes, Gianisa Adisaputri, Richard Torkar, Vladimir Kovalenko, Marcos Kalinowski, Nicole Novielli, Shin Yoo, Xavier Devroey, Xin Tan, et al. 2020. Pandemic programming. *Empirical Software Engineering* 25, 6 (2020), 4927–4961.

[49] David I. Samudio and Thomas D. LaToza. 2022. Barriers in front-end web development. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 1–11.

[50] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*. Springer, 386–400.

[51] Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyou Jiang, Nan Niu, and Qing Wang. 2021. A first look at developers' live chat on gitter. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 391–403.

[52] Makoto Shiraishi, Hironori Washizaki, Yoshiaki Fukazawa, and Joseph Yoder. 2019. Mob programming: A systematic literature review. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Vol. 2. IEEE, 616–621.

[53] Edward Smith, Robert Loftin, Emerson Murphy-Hill, Christian Bird, and Thomas Zimmermann. 2013. Improving developer participation rates in surveys. In *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 89–92.

[54] Diomidis Spinellis. 2005. Version control systems. *IEEE Software* 22, 5 (2005), 108–109.

[55] Diomidis Spinellis. 2012. Git. *IEEE Software* 29, 3 (2012), 100–101.

[56] Michael Stein, John Riedl, Sören J. Harner, and Vahid Mashayekhi. 1997. A case study of distributed, asynchronous software inspection. In *Proceedings of the 19th International Conference on Software Engineering*. 107–117.

[57] Igor Steinmacher, Ana Paula Chaves, and Marco Aurélio Gerosa. 2013. Awareness support in distributed software development: A systematic review and mapping of the literature. *Computer Supported Cooperative Work (CSCW)* 22, 2 (2013), 113–158.

[58] Chengzheng Sun and Clarence Ellis. 1998. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*. 59–68.

[59] Xin Tan, Minghui Zhou, and Li Zhang. 2023. Understanding mentors. engagement in OSS communities via google summer of code. *IEEE Transactions on Software Engineering* 49, 5 (2023), 3106–3130.

[60] Lehana Thabane, Jinhui Ma, Rong Chu, Ji Cheng, Afisi Ismaila, Lorena P Rios, Reid Robson, Marroon Thabane, Lora Giangregorio, and Charles H Goldsmith. 2010. A tutorial on pilot studies: The what, why and how. *BMC Medical Research Methodology* 10, 1 (2010), 1–10.

[61] M. Rita Thissen, Jean M. Page, Madhavi C. Bharathi, and Toyia L. Austin. 2007. Communication tools for distributed software development teams. In *Proceedings of the 2007 ACM SIGMIS CPR Conference on Computer Personnel Research: The Global Information Technology Workforce*. 28–35.

[62] Zhiyuan Wan, Xin Xia, Ahmed E Hassan, David Lo, Jianwei Yin, and Xiaohu Yang. 2018. Perceptions, expectations, and challenges in defect prediction. *IEEE Transactions on Software Engineering* 46, 11 (2018), 1241–1266.

[63] April Yi Wang, Anant Mittal, Christopher Brooks, and Steve Oney. 2019. How data scientists use computational notebooks for real-time collaboration. *Proceedings of the ACM on Human–Computer Interaction* 3, CSCW (2019), 1–30.

[64] Laurie Williams, Robert R. Kessler, Ward Cunningham, and Ron Jeffries. 2000. Strengthening the case for pair programming. *IEEE Software* 17, 4 (2000), 19–25.

[65] Pornpit Wongthongtham, Elizabeth Chang, and Tharam Dillon. 2007. Multi-site distributed software development: Issues, solutions, and challenges. In *International Conference on Computational Science and Its Applications*. Springer, 346–359.

[66] Jin Zhang. 2018. An investigation of technology design features for supporting real-time collaborative programming in an educational environment. (2018). *Master Thesis. The Pennsylvania State University*.

[67] Thomas Zimmermann. 2016. Card-sorting: From text to themes. In *Perspectives on Data Science for Software Engineering*. Elsevier, 137–141.