What's (not) Working in Programmer User Studies?

MATTHEW C. DAVIS, Carnegie Mellon University, USA
EMAD AGHAYI, George Mason University, USA
THOMAS D. LATOZA, George Mason University, USA
XIAOYIN WANG, University of Texas at San Antonio, USA
BRAD A. MYERS, Carnegie Mellon University, USA
JOSHUA SUNSHINE, Carnegie Mellon University, USA

A key goal of software engineering research is to improve the environments, tools, languages, and techniques programmers use to efficiently create quality software. Successfully designing these tools and demonstrating their effectiveness involves engaging with tool users — software engineers. Researchers often want to conduct user studies of software engineers to collect direct evidence. However, running user studies can be difficult, and researchers may lack solution strategies to overcome the barriers, so they may avoid user studies. To understand the challenges researchers face when conducting programmer user studies, we interviewed 26 researchers. Based on the analysis of interview data we contribute: (i) a taxonomy of 18 barriers researchers encounter; (ii) 23 solution strategies some researchers use to address 8 of the 18 barriers in their own studies; and (iii) 4 design ideas, which we adapted from the behavioral science community, that may lower 8 additional barriers. To validate the design ideas, we held an in-person all-day focus group with 16 researchers.

CCS Concepts: • **Software and its engineering**; • **Human-centered computing** → **User studies**;

Additional Key Words and Phrases: empirical software engineering, user study, meta study, human participants, research methodology, human subjects, experiments

1 INTRODUCTION

Research in software engineering often intends to help a programmer work better in some way. For example, fault localization tools [29, 50, 74, 103] may have the goal of helping programmers identify the cause of defects more effectively. Documentation tools [58, 59, 64] may seek to help programmers share or gain knowledge more effectively. The goal of domain-specific programming languages [35, 69, 75, 105] may be to make programming more natural and/or expressive for domain experts. In these cases, the aim is to improve software development by enabling a human programmer to work more effectively.

Will programmers benefit from a new tool, language, or feature? Are our existing tools, languages, or features any good? Answering questions like these may require a **programmer user study** to collect direct evidence while human programmers use a tool, language, or feature. Unfortunately, researchers looking to conduct studies with programmers face a number of significant barriers. A survey of paper authors at OOPSLA, ICSE, CHI, FSE, and other venues found that 84% agreed with the statement that "user evaluation is difficult" [21]. Barriers

Authors' addresses: Matthew C. Davis, Carnegie Mellon University, Institute for Software Research, Pittsburgh, Pennsylvania, USA, mcd2@cs.cmu.edu; Emad Aghayi, George Mason University, Computer Science Department, Fairfax, Virginia, USA, eaghayi@gmu.edu; Thomas D. LaToza, George Mason University, Computer Science Department, Fairfax, Virginia, USA, tlatoza@gmu.edu; Xiaoyin Wang, University of Texas at San Antonio, Department of Computer Science, San Antonio, Texas, USA, xiaoyin.wang@utsa.edu; Brad A. Myers, Carnegie Mellon University, Human-Computer Interaction Institute, Pittsburgh, Pennsylvania, USA, bam@cs.cmu.edu; Joshua Sunshine, Carnegie Mellon University, Institute for Software Research, Pittsburgh, Pennsylvania, USA, sunshine@cs.cmu.edu.

1049-331X/2023/3-ART
https://doi.org/10.1145/3587157

reported included recruiting participants, the necessary time required, and the practical knowledge needed to design and conduct a user study [21].

As a result, conducting programmer user studies remains infrequent among software engineering researchers. An ESE 2015 systematic literature review of tool evaluations published in ICSE, FSE, TSE, and TOSEM from 2001 to 2011 [54] found that while 82% of papers described a tool, only 17% included an empirical evaluation with a human. Of these 17%, over half reported the authors' own experience using the tool. The remaining 83% did not evaluate tool utility with users or relied entirely on *indirect* evidence, such as by evaluating precision and recall of algorithms.

While some individual researchers adopt practices and build infrastructure to reduce their own barriers to conducting studies, this knowledge and infrastructure is rarely shared beyond their research group or local community. The goal of our current work is to identify barriers and disseminate barrier-lowering solution strategies some researchers employ that are not well-known or widely-adopted. These solutions strategies may encompass practice, infrastructure, or both.

Our research intends to answer three variants of the same question about programmer user studies: *"What barriers do researchers experience in X for programmer user studies, and what practices or infrastructure might help reduce these barriers?"* The three values of 'X' are:

(1) X = **Recruiting** participants (this **R**esearch **Q**uestion will be abbreviated $\mathbf{RQ}_{recruiting}$)
(2) X = the **Effort** required ($\mathbf{RQ}_{effort}$)
(3) X = the **Knowledge** required ($\mathbf{RQ}_{knowledge}$)

These questions mirror the research of past studies [21] yet are general enough that they elicited unexpected barriers in our interviews.

To begin to answer these questions, we interviewed 26 researchers about their experience running programmer user studies and used inductive thematic analysis [17] to analyze the interview data and address the research questions. The results contribute: a **taxonomy of 18 barriers** researchers report encountering (Table 4), **23 solution strategies** researchers report using to lower 8 of the 18 barriers (Table 5), and **4 community infrastructure design ideas** (Table 8) adapted from the behavioral science community [3, 25, 77] that may lower 8 additional barriers. Addressing the final 2 barriers remains future work. To validate the 4 design ideas, we held an in-person all-day focus group with 16 researchers, who provided context and insights that are reflected in Section 6.

2 BACKGROUND

This research focuses on task-based experiments where programmers are the participants. Examples include A/B Testing, Exploratory Lab Studies, and Rapid Prototype Evaluations. There are relevant questions and barriers for *other* types of studies (e.g., Contextual Inquiry, Survey, Data mining) we did not ask about. While our findings may apply to these *other* types of studies, we do not attempt to analyze that here.

Each task-based programmer user study has various stages in its development and execution. Figure 1 expands Ko et al. 2015's "canonical experiment" diagram [54] to include *planning*, which Ko et al. 2015 describes but omits from its diagram for reasons that are appropriate for that paper. Below we describe the four stages shown in Figure 1:

(1) **Planning**. Prior to conducting human study research, planning and approvals are generally required. Key steps for the researcher include: learning the background knowledge needed for the study, choosing research questions, selecting study methods and protocols, planning recruitment, designing tasks, identifying data to collect, planning the analysis of the data, getting approvals, etc. These interrelated steps often require multiple iterations.

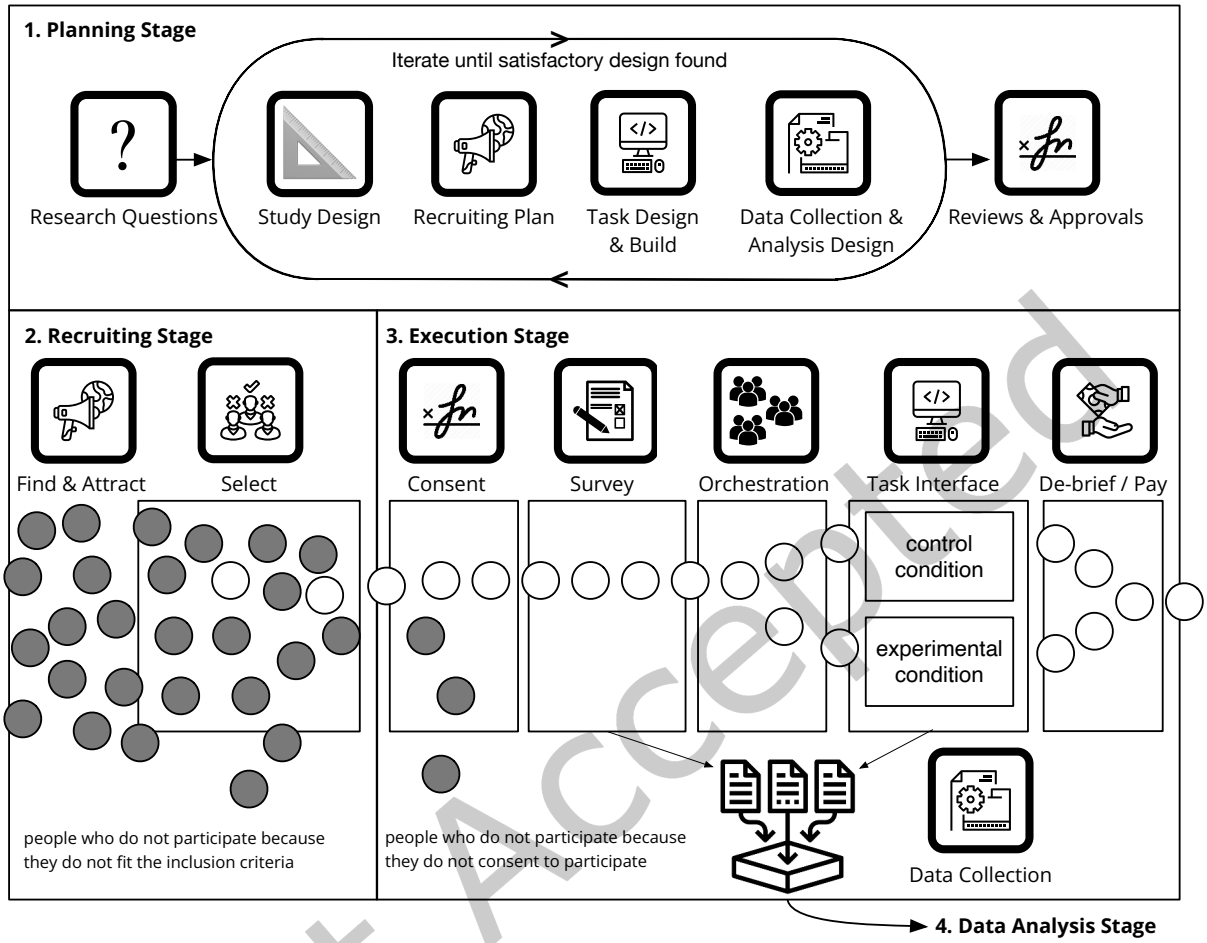

Fig. 1. Key stages in a programmer user study. Within each stage, activities may occur in various orders. This is an augmented version of Figure 1 of Ko et al. 2015 [54]

.

(2) **Recruiting**. Participants must be found and recruited into the study. For many studies, these participants must be representative of the intended user population.
(3) **Execution**. Data must be collected from recruited participants while guiding them through the study steps, which may include: obtaining participant consent, collecting demographic data, assigning the participant to a group, presenting tasks, debrief, and payment (if offered).
(4) **Data Analysis**. Analyzing the data collected to answer the research questions and choosing if or how to disseminate and package the findings.

Throughout this paper, we use several definitions. We inclusively define **programmer** as anyone who writes programs, whether as an end-user, professional, or student. This definition reflects the diversity of how our participant researchers define programmers in their own studies. We differentiate **programmer tools**, which encompass anything used by a programmer to develop software [55], from **study tools**, which facilitate a

researcher's management and execution of a study. Consider a study where a programmer writes code in a novel language using Visual Studio while being observed via Zoom. Visual Studio and the novel language are "programmer tools" and Zoom is a "study tool." Throughout this paper, we adopt Shadish et al.'s [78] definition of **external validity** as "validity of inferences about whether the cause-effect relation holds over variation in persons, settings, treatment variables, and measurement variables" and **internal validity** as "the validity of inferences about whether observed covariation between *A* (the presumed treatment) and *B* (the presumed outcome) reflects a causal relationship from *A* to *B* as those variables were manipulated or measured."

3 RELATED WORK

Programmer user studies are not new: Zendler [113] reports that Grant 1968 [40] is the first programmer user study in the literature. One of the earliest Human-Computer Interaction books was published in 1971 [104] and was about the study of programmers.

As researchers attempted to interpret and use early study results, barriers became evident. Basili et al. 1986 [9] reviewed the early study papers and identified several barriers such as: insufficient planning and motivation of studies, unclear presentation of results, and vast differences in both programmer performance and environments. A framework and careful presentation of results were proposed as solutions. Basili 1993 [7] observed the insufficient *quantity* of experiments and provided advice for researchers. Basili 1996 [8] argued that software engineering needed to improve through well-designed experiments, similar to other scientific disciplines.

Over a twenty one year period, researchers identified further barriers and proposed various solutions using papers as an underlying data source. For example:

- Pfleeger et al. 1995 [67] addressed an implicit barrier of missing knowledge by proposing and describing in detail a set of experiment stages (conception, design, preparation, execution, analysis, dissemination and decision-making), which are similar to our activities in Figure 1, except we draw stage boundaries based on how interviewees talked about their studies.
- Kitchenham et al. 2002 [53] observed many studies would be unsatisfactory if assessed using guidelines from other fields (e.g., medicine); this paper proposed guidelines for software engineering researchers and reviewers to improve the quality of studies.
- Sjøberg et al. 2002 [87] and 2005 [91] observed past studies used insufficiently-realistic tasks, participants, or environments, which may not support external validity. The authors propose that researchers should run more complex, expensive experiments and demand the resources needed to achieve sufficient realism.
- Buse et al. 2011 [21] surveyed papers to show the number of user studies in the literature had increased in absolute and relative terms, and that papers containing user studies enjoyed positive benefits. The authors surveyed researchers and provided early insights that: (i) many researchers planning and executing user studies experience barriers, and (ii) the set of barriers researchers experience may differ from the set of barriers observable in the literature alone.
- Ko et al. 2015 [54] surveyed papers and continued to observe a scarcity of user studies. Informed by this reality and by Buse et. al.'s [21] survey data, the authors provide detailed and practical advice for researchers conducting programmer user studies.

A literature-focused approach has clearly been impactful by directing attention to important and widespread barriers and by providing an opportunity for experts to propose reasonable and informed solutions that aid the community's efforts to improve the quality of its knowledge.

But as Basili 1993 [7] noted, the *quantity* of studies is also of concern. This is reasonable: user studies contribute important *direct* evidence to the community's body of knowledge. In 2007, Sjøberg et al. [90] estimated that the quantity of user studies may be deficient relative to needs by an order of magnitude. Buse et al. 2011 [21] reported

Table 1. Qualitative Research Methods Used with Researcher Participants

	Semi-structured Interviews				Focus Group
	Individual	**Group 1**	**Group 2**	**Group 3**	
Researchers	18	4	4	4	16
Duration	30 minutes	90 minutes			All-day
Method	Inductive Thematic Analysis				Focus Group
Delivery	Remote via Zoom				In-person
Discussed in	Section 5				Section 6

the number of user studies had increased in relative and absolute terms. But this observed increase is insufficient to close the gap, and Ko et al. 2015 [54] continued to find user studies are remarkably "rare."

Given their importance, why are there still so few programmer user studies? What barriers are researchers encountering when planning and running programmer user studies? Why are some researchers struggling, for example, with recruiting while others say they are not? Is one group simply painting a pleasant picture? What is different among these groups?

To understand what is going on, different methods would be valuable: directly talking to people is recommended [11, 70, 71, 81] to understand complex and nuanced questions such as *why* researchers are not publishing more programmer user studies and *why* researchers are experiencing barriers. We therefore talked to researchers who are running programmer user studies.

4 METHOD

Table 1 provides an overview of the methods used in our qualitative study. To gather data about barriers and solutions researchers encounter in programmer user studies, we interviewed 26 researchers, who published a programmer user study at ICSE, CHI, FSE, ASE, or OOPSLA between 2019 and 2021. As demographic data is unavailable for the target population of programmer user study researchers, we selected researchers representing a diverse range of study areas (e.g., SE, PL, ESE, HCI, etc.), experience levels, genders, and roles. We originally planned to interview participants in-person; consequently, many of the participants are located in North America. Researchers were recruited into the study via e-mail. Due to the pandemic, interviews were shifted to Zoom. We classified the 26 researchers based on role (Faculty, Ph.D Student, Industry), geography, study areas, and years of experience running user studies (<5, 5-10, and 10+) using public information. We conducted 18 individual semi-structured interviews and three group semi-structured interviews of four researchers each using open-ended questions. As shown in Table 2, four researchers participated in both a group and an individual interview. The third author conducted 10 of the individual interviews, and the first author conducted eight. The third, fourth, and sixth authors each conducted one group interview. Quotes from researcher participants are anonymously attributed as **R#**. Later, we recruited 16 researchers (Table 3) into an in-person all-day focus group where we discussed ideas for community infrastructure. We reimbursed reasonable travel expenses for focus group participants. Otherwise, we did not compensate any of the participants.

We analyzed interview transcript data qualitatively using the inductive thematic analysis procedure described by Braun and Clarke [17], which calls attention to Frith and Gleeson [39] as a "particularly good example of an inductive thematic analysis." We used Frith and Gleeson as a model for our interview data analysis. While the basis of Frith and Gleeson's data was a questionnaire, we found the described method transferable to interview transcript data.

Table 2. Researcher Interview Participants

Group	ID	Role	User Study Experience	Gender	Geography
Individual Interviews					
Individual	R3	Faculty	5-10 years	Male	North America
	R5	Faculty	10+ years	Female	North America
	R10	Industry	5-10 years	Male	North America
	R11	Faculty	5-10 years	Female	North America
	R13	Faculty	10+ years	Female	North America
	R14	Faculty	5-10 years	Male	North America
	R15	Faculty	5-10 years	Female	North America
	R16	Faculty	5-10 years	Female	North America
	R17	Industry	10+ years	Female	North America
	R18	Faculty	10+ years	Male	North America
	R19	Ph.D Student	< 5 years	Female	Europe
	R20	Ph.D Student	< 5 years	Male	North America
	R21	Ph.D Student	< 5 years	Female	North America
	R22	Ph.D Student	< 5 years	Male	Europe
	R23	Ph.D Student	< 5 years	Male	North America
	R24	Faculty	5-10 years	Male	North America
	R25	Ph.D Student	< 5 years	Male	North America
	R26	Ph.D Student	< 5 years	Male	North America
Group Interviews					
Group 1	R1	Faculty	5-10 years	Female	North America
	R2	Faculty	10+ years	Male	North America
	R3	Faculty	5-10 years	Male	North America
	R4	Post-doc	5-10 years	Male	North America
Group 2	R6	Faculty	5-10 years	Female	North America
	R7	Faculty	5-10 years	Female	North America
	R10	Industry	5-10 years	Male	North America
	R18	Faculty	10+ years	Male	North America
Group 3	R8	Faculty	5-10 years	Male	North America
	R9	Post-doc	5-10 years	Male	North America
	R11	Faculty	5-10 years	Female	North America
	R12	Faculty	5-10 years	Female	North America

Braun and Clarke [17] describes six phases and takes care to emphasize: (i) the phases are guidelines, not rules; (ii) thematic analysis is "not linear" and movement back and forth between phases is expected; and (iii) the process should not be rushed. While we describe the phases below in a linear fashion, we frequently moved back and forth between phases as we tested candidate themes against the data and vice-versa.

Table 3. Researcher In-Person Focus Group Participants

Role	User Study Experience	Gender	Geography
Faculty	5-10 years	Female	North America
Post-doc	5-10 years	Male	North America
Faculty	5-10 years	Female	North America
Industry	5-10 years	Male	North America
Faculty	5-10 years	Female	North America
Faculty	5-10 years	Male	North America
Faculty	5-10 years	Female	North America
Faculty	5-10 years	Male	North America
Faculty	10+ years	Male	North America
Student	<5 years	Female	North America
Faculty	5-10 years	Female	North America
Faculty	5-10 years	Male	North America
Faculty	5-10 years	Male	North America
Faculty	10+ years	Female	North America
Faculty	10+ years	Male	North America
Faculty	5-10 years	Male	North America

- **Data familiarization.** We automatically transcribed each session using Zoom, checked the transcripts against the recordings for accuracy, and made corrections when necessary. An intended effect of this process was becoming familiar with the underlying transcript data.
- **Initial coding.** Interview transcripts were read carefully and reviewed by a researcher to identify timestamps relevant to the research topic and to assign a code that represents a researcher-reported barrier. Timestamps for parts of the transcript involving co-authors or unrelated to the research topic (e.g., the weather) were assigned a null code.
- **Finding themes.** Timestamps with the same code (dealing with the same barrier observation) were grouped together into themes, including solution strategies researchers reported as applicable to a barrier. Each timestamp could be coded to more than one theme as multiple barriers could be discussed at a given timestamp.
- **Reviewing themes.** We systematically reviewed the data to ensure each theme was clearly defined relative to the other themes and was supported by several timestamps and multiple researchers within the underlying transcript data.
- **Defining and naming themes.** We defined the essence of each theme based on the underlying data and named each of the barriers (Table 4) and solution strategies (Table 5) accordingly. Going beyond the recommendations of Braun and Clarke [17] and Frith and Gleeson [39], a separate co-author established replicability by re-coding one group interview transcript and the first and last individual researcher interview transcripts. This resulted in a high level of inter-rater reliability ($K = 0.986, SD = 0.013$).
- **Producing report.** This final step of the process is writing a report, which is this paper.

5 MANY BARRIERS, SOME STRATEGIES

Our inductive thematic analysis of the interviews with 26 researchers identified 18 barriers researchers encountered (Table 4) designing and running task-based programmer user studies as well as 23 solution strategies (Table

Table 4. Barriers Researchers Encountered and Strategies Researchers Used

ID	Barrier Description	Research Question	Solution Strategies
1. Planning Stage			
B1	Task design is hard	RQ_{effort}	†
B2	Difficult to understand design trade-offs in advance	"	S1-S5
B3	Can't take any study off the shelf	"	†
B4	IRB requires effort that seems unnecessary	"	
B5	Building and integrating tools is challenging	"	S6,S7
B6	Difficult to get data collection right	"	S6,S8
B7	Lack of knowledge	$RQ_{knowledge}$	S9
B8	Gaining the needed knowledge is inefficient	"	†
B9	Some researchers uncomfortable with people	"	S10
2. Recruiting Stage			
B10	Hard to recruit enough representative participants	$RQ_{recruiting}$	S11-S21
B11	Hard to manage participants over time	"	†
B12	Recruiting material norms vary by study/org.	"	
3. Execution Stage			
B13	Hard to select participants with the desired characteristics	RQ_{effort}	†
B14	End-to-end orchestration is cumbersome	"	S22
B15	Prototype software isn't ready for deployment	"	S23
B16	Deploying to a participant's local PC is challenging	"	†
B17	Deploying to hosted VMs is challenging	"	†
B18	Deploying to the web is challenging	"	†
4. Data Analysis Stage			
	No barriers were reported by participant researchers for this phase		

†=Potential Solution Strategy Design Ideas Discussed in Section 6

5) researchers told us they employed to lower 8 of the 18 barriers for themselves. We expected to observe similar problems and solution strategies across most researchers. Instead, we observed unevenness: solution strategies may be developed and shared within a research group to lower its own barriers, but other research groups may be unaware of these strategies. The following sections are organized by study stage (Figure 1) and discuss both the barriers researchers encountered and the solution strategies some told us they adopted to lower these barriers within their own work. These strategies included both practice and infrastructure innovations.

5.1 Planning Stage

Aspects of a study require planning prior to the study's execution. In this section we discuss the barriers and solution strategies researchers reported during the planning stage.

5.1.1 Task design is hard (B1).

Table 5. Solution Strategies Reported by Researchers, by Barrier

ID	Solution Strategy Description	Sources
Barrier B2: Difficult to understand design trade-offs in advance		
S1	Pilot remote studies in-person	R25,R26
S2	Ask: Was the task realistic?	R16,R21
S3	Reuse task designs with known properties	R14,R19
S4	Consider using "found" tasks	R21,R22,[54]
S5	Measure remote participant attention	R19,R22,[44]
Barrier B5: Building and integrating tools is challenging		
S6	Reusing tools with known properties (in general)	R13,R14,R19,R23,R25
S7	Evaluate new tools early and using lightweight methods	[21, 23, 55]
Barrier B6: Difficult to get data collection right		
S6	Reusing tools with known properties (for data collection)	R14,R19,R26
S8	Collect data manually	R5,R10,R21,R23,R26
Barrier B7: Lack of knowledge		
S9	Utilize experts, professional network, and the literature	R19,R25
Barrier B9: Some researchers are uncomfortable working with people		
S10	Start with online studies if uncomfortable with human subjects	R12,R22
Barrier B10: Hard to get enough representative participants		
S11	Recruit students when students are known to be representative	R3,R4,R11,R15,et al.
S12	Build and use a personal network on LinkedIn	R10,R13,R16,[27, 54]
S13	Leverage influential Twitter community members	R16,R19,R20
S14	Announce on Reddit, but follow the rules	R18,R19,R21,R24
S15	Use online marketplaces with caution	R20,R22,R23,R25,[56, 95]
S16	Partner with or intern at a research-friendly company	R10,R15,R17,[54, 88, 90]
S17	Deliver the study remotely to maximize recruitment pool size	R11,R19,R2,R24,[20, 54]
S18	Make the recruitment request informal and personal	R10,R11,R17
S19	Emphasize connection in the recruitment request	R13,R15,R17
S20	Explain the wider benefit in the recruitment request	R10,R16,
S21	Offer compensation	R3,R10,R23
Barrier B14: End-to-end orchestration is cumbersome		
S22	Automate parts of the study	R14,R19,R20,R22
Barrier B15: Prototype software isn't ready for deployment		
S23	Minimize deployment environment variability	R10,R13,R18

Not all solution strategies will apply in all situations

> *"We wanted to do a really realistic task. And then we realize, yeah, well, we can't ask somebody to spend, like, two days programming, you know, this task." (R18)*
>
> *"[...] you want [the participant] to do a relatively straightforward programming task, but then coming up with the actual task itself is where I end up with a lot of trouble anyways." (R25)*
>
> *"Of course, like, the actual hardest part of the study was picking the bugs to introduce into those programs [...]" (R26)*

Researchers who plan and execute task-based programmer user studies told us tasks are hard to design. That is not surprising: task design often involves inventing a solution that satisfies diverse requirements. Among the possible task design requirements, some are shown in Table 6. Designing a task to satisfy a single requirement is difficult; designing a task that satisfies *all* the requirements is particularly difficult. For instance, lengthy or difficult tasks may lead to participant fatigue and affect the validity of the results. Hence, tasks must be limited in length and difficulty [54].

Task design often involves trade offs [83, 87]. For a task to support external validity, it must be realistic. Similarly, for a task to support internal validity, it must be constrained such that the effect under study is measured accurately and not confounded by other variables. These needs are at odds [87], and the researcher must find a balance appropriate to the research question, as described by Siegmund, Siegmund, and Apel [83].

Some task requirements represent a linkage among the task design and other aspects of the study that must be satisfied. For instance, a task design must generate data that may be analyzed by the researcher to answer the research question(s); consequently, "measurable" represents a linkage between task design, data collection, data analysis, and the research questions the study is intended to answer. The connections among task design and other aspects of the study increases the difficulty of task design such that a change to other aspects of the study may necessitate changes to the task design. Similarly, if changes within a task impair its ability to satisfy the study's needs, other aspects of the study design may require adaptation. In this way, changes may propagate within the study. For example, an in-person study of a debugging aid may satisfy all requirements and provide an easy way for researchers to observe participant attention and confusion. However, in-person recruiting challenges and a power analysis might convince the researcher to deliver the study remotely to have access to a sufficient number of participants. Moving to remote delivery requires the researcher to decide how to check for attention and confusion within the task. If the researchers decide to use Zoom and the participant's camera to monitor for attention and confusion, additional approval from the Institutional Review Board (IRB) may be necessary as well as a revision to the consent protocol. Further, the researcher must determine how to provide the debugging

Table 6. Some of the Requirements a Task Design May Need to Satisfy

Requirement	Description
Realistic	Resembles a task the target audience may perform outside the study
Constrained	Controls variability such that the expected benefit may be measured
Measurable	Provides data suitable to measure the benefit under study
Achievable	Not so hard that a participant cannot do it
Brief	Fits within the time available without causing participant fatigue
Understandable	Simple enough for the participant to understand
Approachable	Not require knowledge the participant does not have
Propensive	Highly likely the novel feature or tool will be used by the participant
Demonstrative	Highly likely to demonstrates the degree to which a benefit exists
Integrated	The study and developer tools in the task work together

tool and environment to the now-remote participant, as discussed in Section 5.3.1. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

5.1.2 *Difficult to understand design trade-offs in advance (B2).*

> *"Even as experienced empiricists, we have little understanding of what our particular target audience [will] do with the particular tool we're interested in at that time in that particular kind of task." (R5)*

> *"You're never going to get the study design right on the first try; you have to pilot it." (R20)*

Experienced researchers told us they cannot predict what participants will do in a study. Participants might, for example, misunderstand instructions, get bogged down in an unimportant detail, or work in ways that lead to unexpected variability. Experienced researchers pilot their studies with a small set of participants and collect data to evaluate the study's properties to improve the next iteration of the study. In other words, researchers offered no known shortcut to determine in advance whether a study design meets their needs. However, some researchers use the following strategies to lower the impact of this barrier by reducing the number of pilots and design iterations required to find a satisfactory design.

Strategy S1: Pilot remote studies in-person. Any remote study may be delivered and observed in-person. Researchers we interviewed noted it is easier to identify problems with an in-person study than with a remote study. Two researchers take advantage of these two observations by running early pilots of remote studies in-person to quickly identify confounding factors, e.g., participant confusion, stumbling blocks, variability, and so on. These observations then inform the next iteration of the design.

Strategy S2: Ask: Was the task realistic? Some researchers ask the participant in the post-survey whether the task was realistic, meaning the task was similar to one the participant might perform in their usual work. This strategy provides: (i) direct evidence from the participant as to the task's realism and (ii) an early warning to the researcher if a task is potentially unrealistic.

Strategy S3: Reuse task designs with known properties. Reusing previously-designed tasks with known properties may provide a researcher more certainty as to the range of behaviors participants might exhibit when they encounter the task as well as improve the researcher's understanding of the linkages among the task and other parts of the study design — without needing to first run a pilot. The task design reuse researchers described was within a research group where tooling tends to be more uniform. Differences among research groups' tooling may limit the benefit of this strategy since the software underlying the task design may need to be ported or re-implemented according to the receiving group's tooling. Unlike in behavioral science [3, 25, 77], there is not presently a repository for task designs or common experiment tooling (but see Section 6, where we discuss how this may be achieved). While not directly suggested by our researcher participants, Miller [62] cautions that the bias of a task design should be considered prior to its reuse so as to control the propagation of previous bias into a new study.

Strategy S4: Consider using "found" tasks. Ko et al. 2015 [54] discusses using "found" tasks, such as actual bugs from an actual codebase rather than inventing a new codebase and then inventing new bugs for participants to find and fix. One researcher described the difficulty of finding tasks that are sufficiently brief and not too difficult. Researchers who shared using existing codebases in their user studies said they look for tasks in codebases that are neither too large (overly-difficult) nor too small (unrealistic). One researcher avoids popular codebases to reduce the chance that a participant has worked with the codebase before.

Strategy S5: Measure remote participant attention. Compared to in-person studies, researchers told us it is harder to monitor whether a remote participant is attentive to the task vs. checking their phone, daydreaming, etc. Undetected inattention is a confounding variable that degrades the experiment's internal validity. It is possible to mitigate this via webcam, but this carries trade-offs: e.g., the camera opens the participant's home or office to the researcher, which may not be desired – and may turn off some participants. Alternatives researchers reported

using successfully include: (i) prompts that periodically check that the participant is actively working and paying attention to the task, (ii) using automated data collection within the task by which the researcher may infer the participant's level of attention to the task, and (iii) asking the participant whether they became distracted or had to step away from a task during the post-survey. In the behavioral science literature, Hauser et al. [44] offers guidance similar to what our participants reported.

5.1.3 Can't take any study off the shelf (B3). Replications, reproductions, and adaptations are an important part of the scientific process. Miller [62] observes, "deriving reliable empirical results from a single experiment is an unlikely event." Brooks et al. 2008 [19] states, "the experiment process can be error-prone." For these and other reasons, the need for more software engineering replications is a frequent topic in the literature as are the barriers and complexities that may impede these essential studies [16, 51, 52, 60, 63, 72, 79, 82, 102].

Unlike a piece of software, researchers cannot easily study and modify a study's "source code" and use it as a tool to learn the endemic trade-offs and success factors of certain decisions. Freire et al. 2014 [37] cites the lack of a way to formalize an experiment (e.g., in a study configuration file) as a barrier to replication. Miller [62] further notes, "drawing reliable conclusions from reading an article is a difficult task." Typically, a description of the study is provided in a paper and various artifacts or a lab package may be available, but this may still be an incomplete view of the study and may lack important details from Figure 1 such as recruitment materials or details on how groups were counter-balanced. Further, research-quality software artifacts may suffer from so-called "bit-rot" [98] such that after the study is complete, the underlying software is difficult to get running again, possibly due to assumptions about or evolution of the surrounding software ecosystem. Consequently, the task of learning from an existing study may often be a question of reverse-engineering a study based on the information available rather than picking up a working study and adapting it. These realities make the process of learning from, adapting, or replicating past studies less efficient. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

5.1.4 IRB requires effort that seems unnecessary (B4). Many locales require an IRB or similar organization to approve, monitor, and review research involving human subjects. The amount of effort that IRBs require of researchers may vary by institution, locale, and study. While no researcher we interviewed appeared to question the purpose or role of an IRB, some stated their local IRB's implementation choices introduce more inefficiency and effort than what might be necessary. This barrier is further corroborated by a similar finding by Buse et al. [21].

5.1.5 Building and integrating tools is challenging (B5).

> *"I find myself, you know, asking my students to redevelop a platform for pretty much every study [...]" (R6)*
>
> *"My sense is that every time a PhD student is working on something like [a programmer user study], they end up having to build their own [software] stack [...]" (R13)*
>
> *"[...] if we could just keep the same tool — or just use different tools, but in the same way — it would for sure make all our lives just easier." (R19)*

When a study intends to evaluate the benefits of a novel programmer tool, a prototype of the tool often must be built. Researchers explained that predicting the amount of effort to build these tools is challenging, just like other software projects. Further, researchers told us they are not aware of any best practices or guidelines for creating and deploying research-quality tools. Instead, researchers learn by trial and error and are often under time pressures due to the conference cycle. These pressures crowd out time that might be spent on polishing a tool or making it easily reusable by the wider research community.

Developing the novel tool is just the start: the researcher must make the tool work with other tools in the study to provide a coherent experience to the human participant. Consider a researcher evaluating a new programmer aid, XYZZY, within the Eclipse IDE. Here, a data logging tool such as FLUORITE [107] might collect detailed telemetry data about a programmer's activities. Meanwhile, Zoom might record the screen and audio. Participants in the intervention group will use XYZZY, while participants in the control group will not. While Zoom may be orthogonal in this example aside from synchronizing timestamps, Eclipse, FLUORITE, and XYZZY *must* work together, or integrate, to provide a uniform task experience to the participant.

While examples of tool reuse such as CREXperiment [92] and FLUORITE [107] may be found in the literature, it is hard to find and choose an appropriate tool to reuse because, for instance, there is no central repository in which to find these tools. Further, when a researcher finds a tool that *appears* suitable, it might not integrate easily (or at all) with the other tools in the researcher's software stack. For instance, if XYZZY were written as a Visual Studio Code plugin, FLUORITE would not be compatible as FLUORITE requires Eclipse and is incompatible with Visual Studio Code. The net result of these challenges, as R13 expresses above, is that researchers are expending substantial efforts building (and rebuilding) and integrating (and re-integrating) tools.

Strategy S6: Reusing tools with known properties. In our interviews, we encountered researchers successfully reusing tools developed within their own research groups and applied to multiple studies: researchers told us about off-the-shelf tools such as Zoom (screen share) and Visual Studio Code (IDE) that are in common use. The literature provides evidence of successful reuse of more-specialized tools such as FLUORITE [30, 49, 68, 76, 106–111] and task interfaces such as CREXperiment [18, 93, 94]. But the literature provides many cautions regarding the complexities, trade-offs, and difficulties of reusing artifacts [6, 12, 45, 86, 98]; further, (i) there is presently no centralized or easy way to find tools to reuse, and (ii) even if a tool appears suitable, the effort to understand and integrate a tool into a study's software stack may be significant or infeasible.

Strategy S7: Evaluate new tools early and using lightweight methods. The PLIERS framework proposed by Coblenz et al. [23] provides case studies incorporating user-centered techniques (e.g, Wizard of Oz and Rapid Prototyping) to iterate toward a suitable tool design more effectively than traditional methods. Similarly, LaToza and Myers [55] proposes integrating Human-Computer Interaction Methods into tool development at both the formative and summative stages. Buse et al. [21] also provides evidence that lightweight methods are suitable for effective user evaluation.

5.1.6 *Difficult to get data collection right (B6).*

> *"There was a dedicated programmer on that grant and much of what he did was trying to get logging right. I mean, for five years." (R5)*

> *"[...] you wonder if there was any benefit to actually instrumenting the software because it ends up being pretty easy to just look at the video and go, that was about two minutes for that task." (R10)*

Accurately identifying and measuring cause and effect is important to support a study's internal validity. Data collection in today's studies can be particularly laborious and tedious. While simple measures such as time on task may be gathered by reviewing timestamps of screen recordings, more complex measures, like the number of backtracking steps a programmer took, or the number of files viewed, can require hours of careful review of recordings. Some studies automate measurement by instrumenting the IDE; but beyond FLUORITE [107], this instrumentation is infrequently shared across research groups and requires significant engineering investment to build.

Some researchers told us they avoid automated data collection and instead observe or record the participant so the researcher may log events or collect data manually. The observation that capable, intelligent, experienced, highly-educated software engineering researchers take a "pass" on automated data collection is an important signal of both its limitations and its complexity. Certainly, manual data collection has downsides, among them

being: opportunity for manual error, constraining the scale of the study, consuming research hours that might otherwise be applied to the next study, etc. But manual data collection may bring advantages, such as avoiding the effort to build automated instrumentation and allowing measurements that require understanding programmer intent, which may be infeasible to determine automatically. For example, some past studies attempted to measure how long programmers spend debugging based on debugger log data. But this measurement is inaccurate: the debugger may be used for tasks other than debugging and programmers may debug without using the debugger. Here, human judgement is required. The researcher must evaluate all considerations and choose an appropriate data collection strategy to answer the research question in a way that supports internal validity.

> *"Our instrumentation and our data analysis and our monitoring and all of those things — those don't come for free." (R13)*

> *"I can't tell you how many times I've been through [building logging tools]." (R5)*

For researchers who collect data automatically, why do they build so many data collection tools? A data collection tool may be specific to (i) the research question it helps answer, (ii) the task, and (iii) the developer tools employed. If any of these change, a past data collection tool may no longer be suitable. Given the variety of tasks and developer tools (often custom-built) that researchers employ, it is not surprising that new data collection tools may be needed to support a new study. The connection also runs the opposite way: if a developer tool is incapable of providing the data needed to answer the research question, then either the developer tool or the research question may need to be replaced or adapted.

> *"If you don't have any good way of log processing, it's just a lot of work." (R15)*

Researchers told us about their experiences collecting data from programmer tools. Off-the-shelf IDEs are attractive due to their realism: programmers use these tools in their work today. But APIs provided by the vendor may not be clearly documented or intended to satisfy needs researchers have in mind. This may not become clear until late in the process, i.e., analyzing data from a pilot study. One researcher explained the problem of granularity mismatch. In her case, she needed to understand *what* the programmer was clicking on, but the API provided this data at a lower level of granularity: it simply provided clicks and screen coordinates. Using the API, it was not possible to know *what* the programmer was actually clicking on.

> *"[O]ver the last five years or so any time a new student wants to do a different kind of thing, we just add an extra layer of data collection on top of the same platform so that any user study that's done from then on automatically has whatever random ass data collection needs to be in there." (R14)*

Strategy S6: Reuse tools with known properties. We discussed Strategy S6 in the previous section (B5). It may also apply to this barrier, B6.

Strategy S8: Collect data manually. Manual data collection often requires the researcher to directly observe or record the behavior to be measured and then to code event data. As the number of participants, measures, and measurement complexities increase, so does the level of effort; consequently, this strategy may be most feasible for simpler data collection needs with fewer participants or for those requiring human judgement.

5.1.7 *Lack of knowledge (B7).*

> *"Many software engineers believe they aren't trained for [running user studies] — and they're right." (R5)*

> *"It's just that, like, as part of computer science, we give zero training to students going out there on how to conduct valid studies [...]" (R14)*

> *"The post-doc that I had working with me at that time, [name redacted], had some experience [...], so she could easily guide me through it." (R19)*

Designing and running a programmer user study requires a significant and disparate set of skills that researchers often acquire "on the job." Senior researchers pointed out that new researchers often feel they are not prepared to design and run a user study because typical Computer Science curriculums lack training for human study design, task design, qualitative and quantitative research methods, local IRB protocols, recruiting participants, etc. Even a researcher experienced with one type of study may lack knowledge relevant to other types of studies. Regardless, a researcher requires specific knowledge to successfully design and run a study; otherwise, the resulting study may be invalid or flawed.

Strategy S9: Utilize experts, professional network, and the literature. Early-career researchers told us they overcame this barrier by talking to a colleague or advisor experienced with the desired type of study and by reading suggested papers or books. Relying solely on the literature has limits: key design decisions, norms, and trade-offs are not always clear. One researcher reported a colleague shared pitfalls common to the planned study type and suggested solutions that were integrated at an early stage of design. While not directly mentioned by our researcher participants, consulting with community experts who are familiar with the pitfalls and success factors of a particular type of study is a natural variation of this strategy.

5.1.8 Gaining the needed knowledge is inefficient (B8).

> *"How do you know that you've appropriately found a match between the novelty that you're proposing in your paper and, like, recognized metrics — like, these are really hard questions." (R9)*

> *"And you put some metric in your paper and you had one reviewer say you should have done it this way, and another reviewer says you should have done it this other way." (R11)*

While the previous barrier is concerning the lack of knowledge, this barrier is concerned with the inefficiency of *gaining* the knowledge that is missing. Researchers described as a barrier the lack of access to organized, updated learning resources, and best practices for running programmer user studies. The lack of knowledge organization obscures the community's norms and best practices, which can be confusing to paper authors and reviewers alike. These differences may lead to paper rejections that are costly to a researcher's career. Coupled with the steep learning curve, experienced researchers reported the ramp-up time required for new researchers to become productive is significant. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

5.1.9 Some researchers are uncomfortable working with people (B9).

> *"Like, just the the idea of running [a user study] for some people, [it] just doesn't, it doesn't match for them." (R12)*

Some early-career researchers may be uncomfortable working with human subjects; for example, students experiencing a language or communication barrier or social anxiety may avoid human-focused research at this present stage of their career.

Strategy S10: Start with online studies if uncomfortable with human subjects. One early-career researcher reported telling his advisor, "I want to do a study that has the minimum amount of interaction with [...] humans." This researcher designed and ran a successful online study published at a top venue. Notably, the study was highly automated and required minimal human subject contact. This researcher indicates he plans to run more remote studies in the future.

5.2 Recruiting Stage

To support external validity, a study's participants often must be representative of a population that would use the object under study [87]. A past survey by Buse et al. [21] found over 60% of user study researchers reported recruiting to be a barrier. In our interviews, researchers also repeatedly told us recruiting qualified professional

programmers is a barrier to running programmer user studies. While studies need not always have a large *n* for validity, some study designs require recruiting at least a reasonable number of participants.

"Recruiting is definitely the biggest pain point for me." (R1)

"[Recruiting] is really hard. It always keeps me awake at night." (R19)

5.2.1 Hard to recruit enough representative participants (B10).

"One [challenge] is recruiting a sample that's representative [...]" (R13)

"For the professional populations, just finding enough people's the biggest challenge." (R11)

"[Y]ou can't just get students anymore. It's getting harder and harder to publish that, so recruitment is a big problem." (R15)

No academic researcher told us they encounter significant barriers recruiting students: Sjøberg et al. 2003 [88] calls recruiting university students, "relatively easy." In educational studies, students clearly *are* representative. But researchers reported that students do not behave or perform similarly to professional programmers in *all* settings and on *all* tasks — a perspective supported in the literature [13, 32, 34, 48, 54, 65, 73, 87, 88]. A net effect is that researchers must often recruit professional programmers into their studies.

With few exceptions, researchers we interviewed emphasized the difficulty they experience recruiting a sufficient number of representative professional programmers. Lack of access to professional programmers narrows a researcher's range of achievable and publishable studies by excluding studies that require participants with, for example, significant domain experience, specific technology experience, significant practitioner experience, and so on. This is exacerbated by the difficulties described in the literature of selecting a representative sample of professional programmers [2, 5, 26, 27, 57]. Hence, this is an important barrier to address.

Researchers we interviewed shared the following strategies that help lower this barrier in their own research. Strategies S12 - S15 discuss specific social media platforms and online marketplaces. This is not intended to be an exhaustive list. Over time, platforms vary in functionality and popularity; consequently, future researchers should substitute the specific platforms and marketplaces mentioned below with the ones that are most relevant for their specific time, place, and target population.

Strategy S11: Recruit students when students are known to be representative. In settings and tasks where students are known to be representative participants, researchers may avoid the need to recruit professional programmers. For example, in educational research, students *are* the target population. Otherwise, some researchers recruit a mixture of professionals and students into a study and then may show that professionals and students perform similarly, *in this case*. If students and professionals do not perform similarly, then more professionals must be recruited.

Strategy S12: Build and use a personal network on LinkedIn.

"LinkedIn gives me the metadata to do the sampling that I need to do." (R13)

Ko et al. 2015's [54] systematic review of papers observed most studies recruited via existing relationships. LinkedIn provides access to a network of individuals including the individual's experience, skill, and high-level demographic data — as it is known to LinkedIn. Students eventually become professionals. With each class taught and each student encountered, an academic researcher may encourage each student to connect on LinkedIn and build a large pool of potential professionals over time. A trade-off with this approach is the potential for selection bias as the pool of participants within a researcher's network may not generalize. Further, this method requires time to grow the network. As to the quality of participants recruited via LinkedIn, de Mello et al. 2015 [27] found evidence that Java programmers recruited via LinkedIn may demonstrate more experience than those recruited via Mechanical Turk, although this finding might not generalize to other populations.

Strategy S13: Leverage influential Twitter community members. Some researchers reported success engaging influential individuals in the target community with a large number of followers on Twitter. The

researcher asks via direct message if the community member would retweet the study announcement while explaining why the study might be of interest to their followers.

> *"[I]f you have a controversial title like, 'I bet you can't program this' [...] you can get, like, a lot of attention on Reddit." (R18)*

Strategy S14: Announce on Reddit, but follow the rules. Find the subreddits where target participants interact, follow all subreddit rules, and pre-screen the announcement with the moderators prior to posting. Some moderators may refuse or limit what may be said, e.g., compensation. Check first.

Strategy S15: Use online marketplaces with caution.

> *"I would not feel confident doing further work on [Mechanical Turk]." (R20)*

> *"We found that we couldn't really trust the data [on Mechanical Turk]." (R11)*

Several researchers we spoke to used Mechanical Turk [95] to recruit participants and appreciated its ease of recruitment. But each of these researchers explained the significant effort required to detect and filter many inattentive and unqualified participants, which Ahler et al. [1] and Hauser et al. [44] indicate might be a common problem on this platform. A recent study by Tahaei and Vaniea [96] found Computer Science students were more likely to be qualified than self-reported developers recruited through Mechanical Turk and other marketplaces. de Mello et al. 2015 [27] found evidence that Java programmers recruited via LinkedIn may demonstrate more experienced than those recruited via Mechanical Turk. Ko et al. 2015 [54] and Tahaei and Vaniea [96] mention marketplaces besides Mechanical Turk, but we did not encounter researchers using the specific other marketplaces they mentioned. One researcher interviewed pointed to Lau et. al [56] as a recent successful study utilizing UserTesting.com to recruit remote end-user programmers.

Strategy S16: Partner with or intern at a research-friendly company. Ko et al. 2015's [54] systematic review of papers observed some researchers recruited professional participants via a company insider, via a graduate student intern placed at the company, or by establishing a formal partnership with the company. Two industrial researcher participants corroborate this observation by explaining recruiting attempts targeting programmers inside a company must originate from inside the company to be successful, which helps explain the experiences reported by Baltes and Diehl [5]. Further collaboration options are enumerated by Sjøberg et al. 2003 [88] and 2007 [90].

Strategy S17: Deliver the study remotely to maximize recruitment pool size.

> *"[A remote study] definitely broadens the potential audience." (R26)*

Past research provides evidence that remote studies have fewer barriers to participation [20, 54]. In addition to pandemic safety protocols at the time, greater access to professionals was cited by researchers as an important factor for deciding to run a study remotely.

Strategy S18: Make the recruitment request informal and personal. Some researchers explained they communicate in their own name from their work or academic e-mail address, not a generic or mailing list address, and avoid formal language. One industrial researcher reported a marketing person was assigned to polish research communications to improve response rate, but making recruitment communications more formal and less personal had the opposite effect.

Strategy S19: Emphasize connection in the recruitment request. If the potential participant has a connection with the researcher or institution, some researchers make that connection clear in their request. Network-focused tools such as LinkedIn may make this task easier by emphasizing points of connection in ways platforms such as Twitter, GitHub, and Reddit may not.

Strategy S20: Explain the wider benefit in the recruitment request. While direct benefit to the participant is not necessary, some researchers emphasize the benefit the study may have to the wider community to attract representative programmers into their studies.

Strategy S21: Offer compensation. Researchers shared their perceptions that professional programmers are busy, highly-paid, and expect significant compensation for their time. For instance, Bergesen et al. 2014 [14] notably describes expending €40,000 to recruit 65 professional Java programmers into a multi-day study. In our interviews, some researchers told us compensating programmer participants is a solution strategy that has helped lower this barrier for them, but it should be noted that funding realities may be a limiting factor for many researchers.

5.2.2 Hard to manage participants over time (B11). Researchers told us reusing a pool of participants across multiple studies may be useful, for example, to: (i) ease recruiting for future studies where past qualified participants may be selected for a new study based on known demographic data, and (ii) keep track of past participants so they can be excluded from joining a new study when their experience with a prior study may present a confounding factor. Some industry researchers we interviewed use internal participant pools, which is feasible given their access to employee programmers on the payroll. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

5.2.3 Recruiting material norms vary by study/organization (B12). Researchers explained recruiting materials are study-specific and must: (i) attract *specific* participants to a *specific* study, (ii) elide details that may bias or prime the participant, and (iii) comply with local IRB protocols. Researchers shared no clear solutions to this barrier.

5.2.4 Hard to select participants with the desired characteristics (B13). Some researchers explained that it is difficult to select participants with the desired characteristics upon their entry into the study. Notably, researchers recruiting via LinkedIn or personal networks did not raise this as a barrier, possibly due to sufficient data being available that allow these researcher to pre-filter participants *prior to* their entry into the study. But researchers using Mechanical Turk encountered many participants who appeared to lack the skills or experience they claimed to possess, even when they could pass a screener test. Often these researchers had to exclude participants after-the-fact due to these differences. Researchers shared no clear solutions to this barrier when pre-screening is not possible, but we have some proposed ideas in Section 6.

5.3 Execution Stage

The execution stage involves guiding recruited participants through the various steps and tasks of the study. At each step, data may be collected to inform subsequent execution steps of the study (e.g., the demographic survey may influence the group to which the participant is assigned) or may be saved for subsequent analysis. Typical execution steps are shown in Figure 1.

5.3.1 End-to-end orchestration is cumbersome (B14). During planning, a researcher determines which *tasks* will be presented to which *participants* in which *order*. A common technique is to randomly assign participants into groups (often called "conditions") and to present all participants in a particular group a particular set of tasks. For example, to compare tool A to tool B, a researcher might randomly assign half the participants to group α and half to group β. All participants in group α will perform programming tasks using tool A, and all participants in group β will use tool B. This type of design, commonly referred to as a **between subjects** design, distributes the variation between participants fairly. However, there are many alternative designs. For example, in a **within-subjects** design, all participants use all tools being compared. In some cases, fully random group assignment may not be the best choice, as the researcher may wish to ensure that groups are balanced in terms of a particular independent variable such as experience with a particular programming language, gender, disability, etc.

A participant experiences one aspect of study orchestration in terms of the study steps presented to them. For instance, when a participant enters the study, the first step may be to provide consent. Once consent is provided,

the participant completes the demographic survey. Next, the participant is guided through the appropriate tasks. And so on until all steps are completed. Orchestration may be performed manually by a researcher, automated by a study tool, or some combination thereof.

Beyond what a participant experiences, researchers explained study orchestration includes tasks a participant often does not see, such as: filtering participants based on demographic data, assigning participants to a group (e.g., α or β), assigning tasks according to the study protocol, spinning up and tearing down VMs, remembering to record a live session, collecting data from the task, and mundane administrative tasks such as good record-keeping and securing data.

> *"So the [orchestration tool] doesn't exist, but I think it would be useful to have like an automated workflow that kind of tracked all of this." (R10)*

Researchers interviewed raised the need for an orchestration tool to help manage these activities. Even with small studies, the manual effort to administer the study, its participants, its data, and its tasks can be burdensome relative to a researcher's time budget. Researchers attributed errors to manual mistakes and a lack of orchestration: e.g., addressing e-mails to the wrong participant, forgetting to follow consent protocols, errors in group assignment, forgetting to record a session, throwing data away due to administrative errors, and other unfortunate outcomes. Several researchers we interviewed ran studies automated to such an extent that a particular participants could complete the study without a researcher being present. But generalized orchestration for programmer user studies was rare in our sample.

One researcher interviewed built a generalized study orchestration tool within his own group. While describing the system as "not pretty," the researcher reports: it is useful and "better than nothing" since it helps the research group reuse its prior tools, automates complex or repetitive tasks, and helps avoid some types of costly or embarrassing manual errors. By reusing the same tools, data collection, and task interfaces from study to study, the researcher reports it is easier to run studies *and* compare some data across studies within that research group.

Strategy S22: Automate parts of the study. To be clear, it is likely unrealistic for many research groups to develop their own generalized study orchestration tools. Doing so may require a large development effort. That said, several researchers reported successfully automating individual studies such that they run unattended, which indicates that non-generalized automation is achievable in some cases to avoid errors and reduce effort.

5.3.2 *Prototype software is not ready for deployment (B15).*

> *"If you haven't gotten their environment just right, you might squander a whole session. And so we've mitigated a lot of that risk by often using platforms that we get to control." (R13)*

Prototype software, by nature, is neither finished nor ready for deployment. A prototype might only work in a narrow set of environments, and configurations it assumes may neither be mainstream nor agree with best practice. The prototype instead is intended to explore a problem or solution. Researchers used four methods to get prototype software in front of participants in a task-based study — with mixed results:

(1) **Web Application Server**. The researcher deploys the prototype to a web server or container the researcher controls. Participants access the tool via a web browser.
(2) **Hosted Virtual Machine (VM)**. The researcher deploys the prototype to a VM the researcher controls. Participants access the tool via screen share or a remote access tool.
(3) **Participant PC**[1]. The prototype is deployed directly onto the participant's PC, which the researcher does not control. The participant accesses the prototype from there.
(4) **Researcher PC**. The researcher deploys the prototype to a PC the researcher controls. Participants access the PC in-person or via screen sharing or a remote access tool.

[1]We include in this method a researcher-provided Virtual Machine deployed on the participant's PC.

Researchers explained variability of the deployment environment may cause the researcher to waste time — perhaps an entire session — troubleshooting and recovering from malfunctions.

Strategy S23: Minimize deployment environment variability. Researchers reported lowering this barrier by deploying the prototype into an environment the researcher controls: a Web Application Server, Hosted Virtual Machine (VM), or Researcher PC. A hybrid alternative is to deploy a VM on a Researcher PC, which provides convenient environment control while reducing the management overhead and/or cost of hosted VMs. With a Web Application Server, researchers reported the participant's web browser may be a point of variation: if the prototype only works correctly in Chrome or Firefox, the participant must use the compatible browser.

5.3.3 Deploying to a participant's PC is challenging (B16). Researchers who selected this route uniformly expressed frustration getting prototype software to run reliably on participant PCs. This is not surprising: participant computers may be like snowflakes: each one is unique due to different operating systems, software, patch levels, web browsers, configurations, and so on. Consequently, prototype code and tooling may not work as expected — or at all. Researchers also told us some participants do not want to install prototype code on their PC. One alternative approach is to provide a virtual machine or Docker image file for the participant to run on their own computer. This approach may not be suitable for participants who do not have these technologies locally or who are unfamiliar with their use. Further, individuals with a slow internet connection will experience difficulty downloading large image files, and the researcher may find it time-consuming or frustrating to help the participant install the VM and extract any data stored within the VM after the study completes. Researchers shared no clear solutions to this barrier (except to avoid it by using the other strategies).

5.3.4 Deploying to hosted VMs is challenging (B17). Researchers told us managing hosted virtual machines can require more sophistication than simply deploying to a Researcher PC. Further, we heard that Hosted VMs can be costly and complicated: it is not always clear to researchers how to build environments that are stable, usable, robust, and secure. Some researchers also expressed difficulty getting log data out of VMs. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

5.3.5 Deploying to the web is challenging (B18). Some researchers said deploying a prototype via a Web Application Server is challenging and limiting in important ways; e.g., that web-based coding tools give a poor experience, cannot handle complex needs, or are less realistic for coding tasks. One researcher experienced problems due to a web security vulnerability in a prototype web application. However, we also spoke to researchers who ran successful web-based studies using tools such as Visual Studio Code and Code Sandbox. Researchers shared no clear solutions to this barrier, but we have some proposed ideas in Section 6.

6 MANY UNMET NEEDS REMAIN

Though important, the 23 solution strategies reported by researchers in Section 5 (Table 5) lower fewer than *half* the reported barriers we identified. In this section, we look for additional solution strategies in 10 experiment platforms. We first examine 7 software engineering experiment platforms. However, none of these platforms are widely used. We then broaden our focus to include three widely used behavioral science experiment platforms. From these platforms, we identify 4 design ideas that might be adapted to programmer user studies to lower 8 additional barriers (Table 7).

6.1 Software Engineering Experiment Platforms

Over the past quarter century, the software engineering community built at least seven experiment platforms to address its various needs. Freire et al.'s [38] systematic literature review covering 2002 to 2011 identified

Table 7. Barriers Without a Researcher-Reported Solution Strategy

ID	Barrier Description	Potential Solution Strategies
	Barriers with Solution Strategies in Behavioral Science	
B1	Task design is hard	S27
B3	Can't take any study off the shelf	S26
B8	Gaining the needed knowledge is inefficient	S25
B11	Hard to manage participants over time	S24
B13	Hard to select participants with the desired characteristics	S24
	Barriers More-Specific to Programmer User Studies	
B16	Deploying to a participant's local PC is challenging	S27
B17	Deploying to hosted VMs is challenging	S27
B18	Deploying to the web is challenging	S27
	Barriers That May Not Have Solution Strategies	
B4	IRB requires effort that seems unnecessary	
B12	Recruiting material norms vary by study/org.	

Table 8. Potential Solution Strategies

ID	Potential Solution Strategy Description	Barrier(s) Lowered
S24	Recruit using a shared pool of known participants	B11,B13
S25	Design studies using wizards and guides	B8
S26	Reuse study configuration files	B3
S27	Reuse configurable task interfaces and components	B1,B16,B17,B18

experiment platforms[2] used in the software engineering literature, including: Ginger2 [99], SESE [22], Experiment Manager [47], eSEE [101], and Mechanical Turk[3] [95]. In our review of prior work, we found three additional platforms built since 2011: ARREST T [24], ExpDSL[4] [42, 43], and K-Alpha [85]. All are summarized in Table 9.

Why do researchers not widely use these platforms to conduct programmer user studies? Freire et al. [38] discusses some important limitations of certain platforms, but does not investigate the circumstances of their disuse. As circumstances of disuse may be varied, nuanced, complex, and not documented in the literature, we contacted the authors of the original platform papers (except Mechanical Turk) to understand their first-hand experience concerning the life-cycle of their platform. We were able to reach the authors of all platforms except the oldest one, Ginger2. We provided an early draft of this section to the individuals we spoke to and incorporated their feedback and corrections into this section and Table 9.

[2]We exclude FIRE [61] and VBER [15] as both are conceptual frameworks, neither tools nor platforms
[3]Mechanical Turk [95] is not specific to software engineering but has been used in many software engineering studies
[4]There are two unrelated tools called "ExpDSL" in the 2010s; this is not Freire et al 2013's [36] or 2014's [37] "ExpDSL"

Table 9. Experiment Platforms Described in this Section

Platform Name	Year	Maintained?*	Primarily Sustained By	Phases
Software Engineering Experiment Platforms				
Ginger2 [99]	1999	No	Unknown	3,4
SESE [22]	2002	No	Single lab	1,3
Experiment Manager [47]	2008	No	Single lab	3
eSEE [101]	2008	No	Single lab	1,3,4
ARRESTT [24]	2016	No	Single lab	3
ExpDSL [42, 43]	2016	No	Single lab	1,3,4
K-Alpha [85]	2021	Proof of Concept	Single lab	1,3,4
Behavioral Science Experiment Platforms				
jsPsych [25]	2015	Yes	Open Source Community	1,3
LookIt [77]	2017	Yes	Hosted Platform	1,2,3
Gorilla [3]	2020	Yes	Subscriptions	1,2†,3

*=as of September, 2022; See Figure 1 and Section 2 for Phase Descriptions; †=via integration

- **Ginger2** (1999) [99] We were unable to reach the authors to gain additional context about this platform's life-cycle, but Freire et al. [38] points out that the platform only supports data collection and analysis, and its experiments must follow a pre-determined process.
- **SESE** (2002) [22, 89] conducted remote and on-site programmer user studies at scale for a single lab where the programmer used a local IDE such as Eclipse while the SESE client received commands from and communicated results back to the central platform. SESE was not intended for broad community adoption as it: (i) used a proprietary codebase that restricted its sharing, and (ii) required a skilled operator to manage experiment execution. As there was no community built around the platform, when the individuals responsible for the system left the lab, SESE was no longer maintained or used[5].
- **Experiment Manager** (2008) [41, 46, 47, 112] provided support for High Performance Computing-specific experiments by instrumenting a programmer's locally-installed tools (e.g., Eclipse, Emacs, vi, shell, jUnit, etc.) and uploading the instrumentation data to the central web-based server for analysis at the end of the experiment. Some instrumentation was captured by creating wrapper programs for terminal commands. Plans existed to provide support for further experiment types; however, when the research project came to an end and the individuals that created the system graduated from the host institution, Experiment Manager was no longer used or maintained.
- **eSEE** (2008) [100, 101], pronounced "Easy," integrated several tools, task interfaces, and a body of knowledge into a platform to support empirical software engineering. As the system grew, keeping the evolving tools and task interfaces integrated consumed more time than the research group could provide. While the central integrated platform stopped being maintained, its templates, data collection tools, packages, protocols, etc. continued to evolve and additional tools were created, such as Experiment Factory [31].
- **ARRESTT** (2016) [4, 24] provided support for executing and reproducing experiments in software testing techniques to address problems previously identified in Neto et al. 2015 [28]. Support for human experiments was planned, but evolution stopped and the platform fell into disuse when the individuals responsible left or graduated from the host institution.

[5]Some ideas from SESE were later incorporated into greps.com, a commercial platform for Java skill evaluations [89].

- **ExpDSL** (2016) [33, 42, 43] provided a platform, including an editor, to conduct human experiments using prototype DSLs. The platform was based on the Meta Programming System (MPS) [97] and did not have a community or company sponsorship; consequently, the platform fell into disuse when breaking changes to the MPS platform also broke ExpDSL, and resources were not available to make the necessary updates.
- **K-Alpha** (2021) [84, 85] is a recent prototype under active development that is too new at the time of this writing to evaluate its full life-cycle, evolution, or sustainment.

Creating, sustaining, and evolving an experiment platform is neither trivial nor risk-free. Various phases in the platform's life-cycle may carry certain risks and opportunities, such as:

- **Planning**: No platform may be expected to support every possible experiment; consequently, it is necessary to choose which experimenters and experiment types to support as well as relative priorities among them. ExpDSL, Experiment Manager, and ARRESTT took a narrow approach to the types of experiments supported. Careful attention should be given to existing platforms on which the new platform may be built: breaking changes, instability, or lack of maintenance in the underlying platform may generate an unexpected need for effort. The authors of ExpDSL experienced this problem when updates to MPS broke ExpDSL.
- **Implementation**: The platform authors used a mixture of contract and in-house labor to build the platforms. This phase may be similar to many other development projects.
- **Sustainment and Evolution**: Community-building is often needed to evolve and sustain a platform over time. Absent a sustaining community, a platform's ongoing viability may be fragile such that it easily declines into disuse.

Software engineering researchers are clearly able to build a wide assortment of experiment platforms that satisfy diverse needs but have thus far experienced less success building communities that might support, sustain, and evolve these platforms over longer periods of time. Further, past software engineering experiment platforms lacked support for community-based solution strategies that are now used in behavioral science experiment platforms. We explore these platforms next.

6.2 Behavioral Science Experiment Platforms

Behavioral science studies share many similarities with programmer user studies in terms of experimental design. In at least three cases, behavioral science researchers have built popular experiment platforms with supporting communities and solution strategies that were notable in our focus group discussions with programmer user study researchers. We outline these experiment platforms below:

- **jsPsych** [25] is a web-based behavioral experiment platform that organizes an experiment into a sequence of steps and decisions. Each experiment step presents a researcher-configurable task interface within which the participant may complete a task. The result of each task is recorded and may influence subsequent steps of the experiment. jsPsych provides a set of core task interfaces appropriate for behavioral science, additional task interfaces are provided by the community, and researchers may create custom task interfaces for their own specific needs. jsPsych does not directly aid in the recruiting or data analysis stages; however, jsPsych's distillation of an experiment's execution steps into an experiment configuration file facilitates replication and adaptation of past experiments.
- **LookIt** [77] is a shared online platform on which researchers from different organizations may design and execute online experiments. It provides the ability to define an experiment's steps and decisions and provides researchers a set of configurable task interfaces that may be presented to a participant during the experiment. LookIt is particularly notable for its community-oriented features: e.g., (i) it provides participants the ability to opt-in to a shared participant pool whereby they may participate in future experiments according to a cadence they choose, and (ii) the platform requires a community review of experiments prior to recruitment to ensure LookIt's community standards are upheld.

- **Gorilla** [3] is a shared online platform on which multiple researchers may design and host experiments. Researchers may use its visual tools to configure task interfaces and insert them into complex experiment designs. Notably, Gorilla provides a set of guides for experiment designs that are intended to help researchers design an experiment. Researchers may use Gorilla to share their experiments with other researchers, and the tool is able to ingest many jsPsych experiments. While Gorilla does not directly support recruitment, it provides some integrations with platforms such as Qualtrics and Prolific.

Behavioral science platforms presently lack support for complex, diverse, and realistic programming task interfaces such that participants may write, compile, test, analyze, and debug code. Consequently, researchers in our focus group explained this gap is a barrier to adoption, which partially echoes Freire et al.'s [38] recommendations for software engineering experiment platforms. However, our focus group discussions identified aspects of these platforms that, if adapted, may lower additional barriers encountered by software engineering researchers.

6.3 Potential Solution Strategies

In this subsection, we explore 4 design ideas from jsPsych [25], LookIt [77], and Gorilla [3] that our focus group discussions indicated might be adapted to lower 8 of the 10 programmer user study barriers that presently lack solution strategies. We note that it is beyond the scope of this paper to prescribe whether it might be more advantageous to (i) extend behavioral science platforms to support programmer user studies, (ii) build a community-supported platform specific to conducting programmer user studies, or (iii) take some other approach. We also leave to future work the exploration of other software engineering study types that are not programmer user studies.

6.3.1 Strategy S24: Recruit using a shared pool of known participants. LookIt [77] provides a shared pool of participants that were previously recruited to the platform for a study. This pool includes participant demographic information and respects the participant's willingness to join a future study. Prolific [66] provides a similar shared participant pool for scientific researchers. Some software companies also maintain participant pools to support various types of research. Researchers in our focus group supported this design idea as a means to help lower recruiting barriers and to allow reuse of prior effort to attract hard-to-find populations. But managing a shared pool requires addressing important considerations such as: participant privacy, participant burn-out through over-contact, and ensuring study invitations and the study itself are high-quality and appropriate. Notably, the design decisions of LookIt address many of these considerations, including that participants control their contact state and frequency, controls are in place to protect participant privacy, and a mandatory study peer review makes it more likely that studies will meet community guidelines prior to being released to the participant pool. Adopting a similar strategy may help lower recruiting barriers in programmer user studies:

- **Hard to manage participants over time (B11)** may be lowered using this strategy to provide researchers a shared pool of programmer participants with known demographic information that have participated in past studies.
- **Hard to select participants with the desired characteristics (B13)** may be lowered by providing researchers visibility into the populations available within the pool, including demographic and experience information relevant to a researcher's study.

6.3.2 Strategy S25: Design studies using wizards and guides. Gorilla [3] offers a study configuration wizard that allows researchers to design various studies through a series of prompts, which allows researchers to benefit by exploring unfamiliar experiment designs that may be appropriate to their goals. Guides may provide study-specific front-line guidance to address many typical questions and pitfalls in a normative way that may point to authoritative checklists or literature. In our focus group, researchers found this idea realistic and that adopting a similar strategy may help lower similar barriers for programmer user studies:

- **Gaining the needed knowledge is inefficient (B8)** may be lowered by guiding researchers through the study design process using a series of wizard prompts and guides, which may make the study design process less overwhelming while providing normative guidance appropriate for the researcher's need during the design process. This design approach may reduce the likelihood that a researcher will make errors relative to community norms or need to find an expert for a particular study design simply to suggest literature to read.

6.3.3 Strategy S26: Reuse study configuration files. Basili et al. 1999 [10], Brooks et al. 2008 [19], Shull et al. 2008 [82], and others [63, 79, 102] discuss the need to communicate key details needed for future researchers to replicate, adapt, or understand a study; yet there are many complexities to doing so [79, 80, 102]. jsPsych [25] and Gorilla [3] help address this need in diverse ways. Every jsPsych experiment is encoded in a shareable study configuration file that may be read, copied, or adapted similarly to a program's source code. Notably, jsPsych delegates the representation of complex configuration details within its configuration file to the task interface itself; consequently, jsPsych avoids the need to be aware of all possible details of every possible task interface. Gorilla [3] facilitates sharing a study design using its web-based portal. Freire et al. 2013 [36] and 2014 [37] provide evidence that a study configuration file may support a wide variety of software engineering experiment types. Researchers in our focus group indicated this design idea may realistically lower barriers to sharing study designs and facilitates a researcher's adaptation of, learning from, and/or replication of a previously-conducted study.

- **Can't take any study off the shelf (B3)** may be lowered by encoding each study into a configuration file that describes the aspects needed to adapt and/or repeat the experiment: the participant pool selection and quantity, the demographic survey, logic to assign participants to groups and tasks, the task interfaces and their configurations, as well as data collection and analysis details. While contextual details such as motivation may be described in a companion paper published by the researchers, the experiment configuration file may provide a clear and transparent picture of the study such that a future researcher may take a past experiment off the shelf and either replicate it or adapt it for their own research purposes using the same platform.

6.3.4 Strategy S27: Reuse configurable task interfaces and components. According to Basili et al. 1999 [10], experimentation time and cost may be reduced by reusing artifacts. But the literature enumerates many difficulties, complexities, and trade-offs involved [6, 12, 45, 86, 98]. Artifact reuse is a key feature of jsPsych [25], LookIt [77], and Gorilla [3]: each provides a variety of configurable task interfaces and components. Our focus group indicated that adopting this strategy may reduce the effort needed to set up and operate a working experiment, including items such as consent forms, demographic surveys, and common tools and task interfaces. Evidence suggests that this design idea is applicable to programmer user studies: one research group we encountered uses an in-house generalized orchestration tool with configurable tools and task interfaces that are reused from study to study. Other researchers we interviewed reuse some tools from study to study but are inhibited by the lack of a generalized orchestration framework to facilitates integration and exchange of task interfaces and components with other research groups. Adapting this solution strategy to the unique needs of the programmer user study research community may help lower multiple barriers:

- **Task design is hard (B1)** may be lowered in many cases by a researcher selecting from a shared library of configurable task interfaces appropriate for programmer user studies. While task design will likely remain challenging, our focus group thought this solution strategy may reduce effort by avoiding the build, test, and evaluation effort required for many custom task interfaces. jsPsych [25], LookIt [77], and Gorilla [3] provide significant libraries of task interfaces, and a similar effort within our own community would be necessary to cover a set of realistic environments and programmer tasks.

- **Deploying to hosted VMs is challenging (B17)** may be lowered by encapsulating common VM hosting operations into a reusable task interface component. Our focus group indicated the ability to automatically manage the life-cycle of spinning up a VM, connecting the participant, managing the VM over the course of the study, and the final stages of data extraction and shut down were automation opportunities. VM and cloud providers offer stable APIs for this automation, but these APIs have a significant learning curve. Hence, providing this automation within the orchestration tool relieves researchers of this burden.
- **Deploying to the web is challenging (B18)** may be lowered by providing pre-integrated configurable task interfaces that utilize realistic web-based environments that some researchers are successfully using in their studies today. These may include tools such as Visual Studio Code, Code Sandbox, GitHub CodeSpaces, CRExperiment, and others. At our focus group, the need for realistic web-based task interfaces was a common concern.
- **Deploying to a participant's local PC is challenging (B16).** This barrier reflects the difference between a participant's local PC, which is of varied configuration and state, and the need for prototype software to make assumptions about the environment in which it will be installed. Researchers in our focus group expressed a desire to side-step this issue by lowering the barriers described above.

7 THREATS TO VALIDITY

The researchers we engaged were selected because they are successful programmer user study researchers, as evidenced by their past publications and meaningful contributions to the field. It is possible researchers we did not talk to encountered significant barriers unknown to us. For example, our sampling method excluded researchers who have not yet published their first study. We attempted to mitigate this limitation by including in our sample junior researchers for whom their first study was relatively recent, but it is reasonable that researchers who fail or give up on their first programmer user study may experience a different set of barriers to which our method and data lacks visibility. Our sample of researchers is primarily from North America; it is possible barriers and solution strategies differ across geographies. We lack demographic data about our target population of programmer user study researchers and are unable to describe how our researcher sample relates to the overall population. Our research questions and interview scripts were designed to surface practice and infrastructure needs; other needs, such as changes to community practices at large, may not be addressed because we did not ask researchers about them specifically. However, we did not restrictively define the word "practices," and we observed that some researchers spoke of community practices in their responses. It is possible additional solution strategies exist for barriers that were unknown to our sample of researchers or that did not seem salient to them at the time. The ongoing pandemic during the study period may have caused some researchers, who might otherwise be reluctant to run remote studies, to run remote studies anyway. Due to this shift, it is possible we encountered researchers at a time when they were experiencing a different set of barriers than usual due to their need to actively adapt their studies to an unfamiliar remote delivery method. Inductive thematic analysis requires an active role of the researcher: themes are tested against the data, but do not "reveal themselves" [17]. While we took steps beyond what Braun and Clarke [17] recommend to mitigate the biases of one author, no process can remove all bias. For this reason, the findings supported by thematic analysis are described as a set of barriers encountered by practitioners along with found solutions. We avoid making broader or more-precise claims than the method and data support.

8 CONCLUSIONS

This paper provides guidance and strategies that researchers can use to conduct more — and more impactful — experiments with programmers. To that end, we engaged the programmer user study research community

through a set of human-focused methods that provide a more-nuanced lens than prior work, which was largely based on literature reviews.

Using Inductive Thematic Analysis [17] to analyze transcripts of 26 researcher interviews (Table 2), we found researchers encounter 18 substantial barriers (Table 4), including barriers recruiting participants, the effort required, and the knowledge required to conduct a programmer user study. Of the 18 barriers reported by researchers, we found 8 have at least one solution strategy in use, but these strategies may neither be well-known nor widely-adopted, even if found in prior work. We summarize these 23 solution strategies in Table 5 and contribute them to the community as they may help lower 44% (8/18) of reported barriers.

We further found 8 of the 10 barriers without a solution strategy (Table 7) may be lowered by adopting 4 community infrastructure design ideas previously adopted by the behavioral science community, as evidenced by LookIt [77], jsPsych [25], and Gorilla [3]. We adapt and contribute these design ideas in Table 8, and compare the sustainment models of 10 experiment platforms in Table 9. Building or adapting a system to support these ideas for our community may increase the number of barriers lowered by known solution strategies from 44% (8/18) to 89% (16/18). The final 2 unresolved barriers remain as future work as are finding and addressing barriers we did not elicit.

Is there support among researchers for building or adapting community infrastructure? In our focus group of 16 researchers (Table 3), we observed support for the 4 design ideas (Table 8). Is building community infrastructure realistic? The evidence indicates it may be: the behavioral science community recently built three software-based human experiment platforms, and we found seven examples of software engineering experiment platforms in the literature, although these platforms often appear to have abbreviated lives due to insufficient community-building. Incorporating similar ideas into the programmer user study community by extending behavioral science platforms or by building a platform specifically for programmer user studies with community support may similarly reduce the burden on researchers, increase the amount and proportion of direct evidence available, and advance the field's progress by making programmer user studies easier, more impactful, and more available to software engineering researchers.

ACKNOWLEDGMENTS

We are grateful for the help of all our study participants, particularly those who travelled during the COVID-19 Delta wave to participate in our all-day focus group where safety protocols were maintained. This material is supported in part by NSF grants 2016600, 2016586, and 2016604. Any opinions, findings, or conclusions expressed in this material are those of the authors and do not necessarily reflect those of any of the sponsors.

REFERENCES

[1] Douglas J. Ahler, Carolyn E. Roush, and Gaurav Sood. 2021. The micro-task market for lemons: data quality on Amazon's Mechanical Turk. *Political Science Research and Methods* (2021), 1–20. https://doi.org/10.1017/psrm.2021.57

[2] Bilal Amir and Paul Ralph. 2018. There is no random sampling in software engineering research. In *Proceedings of the 40th International Conference on Software Engineering: companion proceeedings*. 344–345.

[3] Alexander L Anwyl-Irvine, Jessica Massonnié, Adam Flitton, Natasha Kirkham, and Jo K Evershed. 2020. Gorilla in our midst: An online behavioral experiment builder. *Behavior research methods* 52, 1 (2020), 388–407.

[4] An author of "Arrestt: a framework to create reproducible experiments to evaluate software testing techniques". 2022. Personal communication.

[5] Sebastian Baltes and Stephan Diehl. 2016. Worse than spam: Issues in sampling software developers. In *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–6.

[6] Earl Barr, Christian Bird, Eric Hyatt, Tim Menzies, and Gregorio Robles. 2010. On the shoulders of giants. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 23–28.

[7] Victor R Basili. 1993. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*. Springer, 1–12.

[8] Victor R Basili. 1996. The role of experimentation in software engineering: past, current, and future. In *Proceedings of IEEE 18th International Conference on Software Engineering*. IEEE, 442–449.
[9] Victor R Basili, Richard W. Selby, and David H. Hutchens. 1986. Experimentation in software engineering. *IEEE Transactions on Software Engineering* SE-12, 7 (1986), 733–743. https://doi.org/10.1109/TSE.1986.6312975
[10] Victor R Basili, Forrest Shull, and Filippo Lanubile. 1999. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering* 25, 4 (1999), 456–473.
[11] Kathy Baxter, Catherine Courage, and Kelly Caine. 2015. *Understanding your users: a practical guide to user research methods*. Morgan Kaufmann.
[12] Moritz Beller. 2020. Why I will never join an Artifacts Evaluation Committee Again. https://inventitech.com/blog/why-i-will-never-review-artifacts-again/. [Online; accessed 2022-09-25].
[13] Patrik Berander. 2004. Using students as subjects in requirements prioritization. In *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.* IEEE, 167–176.
[14] Gunnar R Bergersen, Dag IK Sjøberg, and Tore Dybå. 2014. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering* 40, 12 (2014), 1163–1184.
[15] Stefan Biffl and Dietmar Winkler. 2007. Value-based empirical research plan evaluation. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 494–494.
[16] Ronald F. Boisvert. 2016. Incentivizing Reproducibility. *Commun. ACM* 59, 10 (sep 2016), 5. https://doi.org/10.1145/2994031
[17] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
[18] Larissa Braz, Enrico Fregnan, Gül Çalikli, and Alberto Bacchelli. 2021. Why Don't Developers Detect Improper Input Validation?'; DROP TABLE Papers;–. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 499–511.
[19] Andy Brooks, Marc Roper, Murray Wood, John Daly, and James Miller. 2008. Replication's role in software engineering. In *Guide to advanced empirical software engineering*. Springer, 365–379.
[20] Anders Bruun, Peter Gull, Lene Hofmeister, and Jan Stage. 2009. Let your users do the testing: a comparison of three remote asynchronous usability testing methods. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1619–1628.
[21] Raymond P.L. Buse, Caitlin Sadowski, and Westley Weimer. 2011. Benefits and Barriers of User Evaluation in Software Engineering Research. *SIGPLAN Not.* 46, 10 (oct 2011), 643–656. https://doi.org/10.1145/2076021.2048117
[22] E Arisholm Carelius, DIK Sjøberg GJ, and Y Lindsjørn. 2002. A web-based support environment for software engineering experiments. *Nordic J. Computing* 9, 4 (2002), 231–247.
[23] Michael Coblenz, Gauri Kambhatla, Paulette Koronkevich, Jenna L Wise, Celeste Barnaby, Joshua Sunshine, Jonathan Aldrich, and Brad A Myers. 2021. PLIERS: a process that integrates user-centered methods into programming language design. *ACM Transactions on Computer-Human Interaction (TOCHI)* 28, 4 (2021), 1–53.
[24] Iaron da Costa Araújo, Wesley Oliveira da Silva, José B de Sousa Nunes, and Francisco Gomes de Oliveira Neto. 2016. Arrestt: a framework to create reproducible experiments to evaluate software testing techniques. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. 1–10.
[25] Joshua R. de Leeuw. 2015. jsPsych: A JavaScript library for creating behavioral experiments in a Web browser. *Behavior Research Methods* 47, 1 (March 2015), 1–12. https://doi.org/{10.3758/s13428-014-0458-y}
[26] Rafael Maiani de Mello, Pedro Correa da Silva, and Guilherme Horta Travassos. 2014. Sampling improvement in software engineering surveys. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 1–4.
[27] Rafael Maiani de Mello, Pedro Corrêa Da Silva, and Guilherme Horta Travassos. 2015. Investigating probabilistic sampling approaches for large-scale surveys in software engineering. *Journal of Software Engineering Research and Development* 3, 1 (2015), 1–26.
[28] Francisco Gomes de Oliveira Neto, Richard Torkar, and Patricia DL Machado. 2015. An initiative to improve reproducibility and empirical evaluation of software testing techniques. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 575–578.
[29] Richard A DeMillo, Hsin Pan, and Eugene H Spafford. 1996. Critical slicing for software fault localization. *ACM SIGSOFT Software Engineering Notes* 21, 3 (1996), 121–134.
[30] Prasun Dewan. 2015. Towards emotion-based collaborative software engineering. In *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering*. IEEE, 109–112.
[31] Paulo Sérgio Medeiros dos Santos and Guilherme Horta Travassos. 2017. Structured synthesis method: the evidence factory tool. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 480–481.
[32] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Software Engineering* 23, 1 (2018), 452–489.
[33] Michael Felderer. 2022. Personal communication.
[34] Robert Feldt, Thomas Zimmermann, Gunnar R Bergersen, Davide Falessi, Andreas Jedlitschka, Natalia Juristo, Jürgen Münch, Markku Oivo, Per Runeson, Martin Shepperd, et al. 2018. Four commentaries on the use of students and professionals in empirical software

engineering experiments. *Empirical Software Engineering* 23, 6 (2018), 3801–3820.
[35] Martin Fowler. 2010. *Domain Specific Languages.* Addison-Wesley Professional.
[36] Marília Freire, Paola Accioly, Gustavo Sizílio, Edmilson Campos Neto, Uirá Kulesza, Eduardo Aranha, and Paulo Borba. 2013. A model-driven approach to specifying and monitoring controlled experiments in software engineering. In *International Conference on Product Focused Software Process Improvement*. Springer, 65–79.
[37] Marília Freire, Uirá Kulesza, Eduardo Aranha, Gustavo Nery, Daniel Costa, Andreas Jedlitschka, Edmilson Campos, Silvia T Acuña, and Marta N Gómez. 2014. Assessing and evolving a domain specific language for formalizing software engineering experiments: An empirical study. *International Journal of Software Engineering and Knowledge Engineering* 24, 10 (2014), 1509–1531.
[38] Marília Aranha Freire, Daniel Alencar da Costa, Edmilson Campos Neto, Tainá Medeiros, Uirá Kulesza, Eduardo Aranha, and Sérgio Soares. 2013. Automated Support for Controlled Experiments in Software Engineering: A Systematic Review (S).. In *SEKE*. 504–509.
[39] Hannah Frith and Kate Gleeson. 2004. Clothing and embodiment: Men managing body image and appearance. *Psychology of men & masculinity* 5, 1 (2004), 40.
[40] E.E. Grant and H. Sackman. 1967. An Exploratory Investigation of Programmer Performance Under On-Line and Off-Line Conditions. *IEEE Transactions on Human Factors in Electronics* HFE-8, 1 (1967), 33–48. https://doi.org/10.1109/THFE.1967.233303
[41] HPCS Development Time Working Group. 2007. Exp. Manager. https://www.cs.umd.edu/~basili/hpcs/index.php_id=17. [Online; accessed 2022-10-05].
[42] Florian Häser, Michael Felderer, and Ruth Breu. 2016. An integrated tool environment for experimentation in domain specific language engineering. In *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*. 1–5.
[43] Florian Häser, Michael Felderer, and Ruth Breu. 2018. Evaluation of an Integrated Tool Environment for Experimentation in DSL Engineering. In *International Conference on Software Quality*. Springer, 147–168.
[44] David Hauser, Gabriele Paolacci, and Jesse Chandler. 2019. Common concerns with MTurk as a participant pool: Evidence and solutions. In *Handbook of research methods in consumer psychology*. Routledge, 319–337.
[45] Robert Heumüller, Sebastian Nielebock, Jacob Krüger, and Frank Ortmeier. 2020. Publish or perish, but do not forget your software artifacts. *Empirical Software Engineering* 25, 6 (2020), 4585–4616.
[46] Lorin Hochstein. 2022. Personal communication.
[47] Lorin Hochstein, Taiga Nakamura, Forrest Shull, Nico Zazworka, Victor R Basili, and Marvin V Zelkowitz. 2008. An environment for conducting families of software engineering experiments. *Advances in Computers* 74 (2008), 175–200.
[48] Martin Höst, Björn Regnell, and Claes Wohlin. 2000. Using students as subjects—a comparative study of students and professionals in lead-time impact assessment. *Empirical Software Engineering* 5, 3 (2000), 201–214.
[49] Constantina Ioannou, Andrea Burattin, and Barbara Weber. 2018. Mining developers' workflows from IDE usage. In *International Conference on Advanced Information Systems Engineering*. Springer, 167–179.
[50] James A Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*. IEEE, 467–477.
[51] Natalia Juristo and Sira Vegas. 2011. The role of non-exact replications in software engineering experiments. *Empirical Software Engineering* 16, 3 (2011), 295–324.
[52] Barbara Kitchenham. 2008. The role of replications in empirical software engineering—a word of warning. *Empirical Software Engineering* 13, 2 (2008), 219–221.
[53] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on software engineering* 28, 8 (2002), 721–734.
[54] Amy J Ko, Thomas D LaToza, and Margaret M Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Empirical Software Engineering* 20, 1 (2015), 110–141.
[55] Thomas D LaToza and Brad A Myers. 2011. Designing useful tools for developers. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. Association for Computing Machinery, New York, NY, 45–50.
[56] Sam Lau, Sruti Srinivasa Srinivasa Ragavan, Ken Milne, Titus Barik, and Advait Sarkar. 2021. Tweakit: Supporting end-user programmers who transmogrify code. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–12.
[57] Valentina Lenarduzzi, Oscar Dieste, Davide Fucci, and Sira Vegas. 2021. Towards a Methodology for Participant Selection in Software Engineering Experiments: A Vision of the Future. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6.
[58] Boyang Li, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Nicholas A Kraft. 2016. Automatically documenting unit test cases. In *2016 IEEE international conference on software testing, verification and validation (ICST)*. IEEE, 341–352.
[59] Mario Linares-Vásquez, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. Changescribe: A tool for automatically generating commit messages. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 709–712.
[60] Jonathan Lung, Jorge Aranda, Steve M Easterbrook, and Gregory V Wilson. 2008. On the difficulty of replicating human subjects studies in software engineering. In *Proceedings of the 30th international conference on Software engineering*. 191–200.

[61] Manoel G Mendonça, José C Maldonado, Maria CF De Oliveira, Jeffrey Carver, Sandra CPF Fabbri, Forrest Shull, Guilherme H Travassos, Erika Nina Höhn, and Victor R Basili. 2008. A framework for software engineering experimental replications. In *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*. IEEE, 203–212.

[62] James Miller. 2000. Applying meta-analytical procedures to software engineering experiments. *Journal of Systems and Software* 54, 1 (2000), 29–39. https://doi.org/10.1016/S0164-1212(00)00024-8

[63] James Miller. 2005. Replicating software engineering experiments: a poisoned chalice or the holy grail. *Information and Software Technology* 47, 4 (2005), 233–244.

[64] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2014. Automatic generation of release notes. In *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*. 484–495.

[65] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, and Matthew Smith. 2020. *On Conducting Security Developer Studies with CS Students: Examining a Password-Storage Study with CS Students, Freelancers, and Company Developers*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376791

[66] Stefan Palan and Christian Schitter. 2018. Prolific. ac—A subject pool for online experiments. *Journal of Behavioral and Experimental Finance* 17 (2018), 22–27.

[67] Shari Lawrence Pfleeger. 1995. Experimental design and analysis in software engineering. *Annals of Software Engineering* 1, 1 (1995), 219–253.

[68] Marcos Antonio Possatto and Daniel Lucrédio. 2015. Automatically propagating changes from reference implementations to code generation templates. *Information and software technology* 67 (2015), 65–78.

[69] Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frédo Durand, and Saman Amarasinghe. 2013. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. *Acm Sigplan Notices* 48, 6 (2013), 519–530.

[70] Robert Rosenthal and Ralph L Rosnow. 2008. *Essentials of behavioral research: Methods and data analysis*.

[71] Herbert J Rubin and Irene S Rubin. 2011. *Qualitative interviewing: The art of hearing data*. sage.

[72] Per Runeson, Andreas Stefik, and Anneliese Andrews. 2014. Variation factors in the design and analysis of replicated controlled experiments. *Empirical Software Engineering* 19, 6 (2014), 1781–1808.

[73] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. 2015. Are students representatives of professionals in software engineering experiments?. In *2015 IEEE/ACM 37th IEEE international conference on software engineering*, Vol. 1. IEEE, 666–676.

[74] Raul Santelices, James A Jones, Yanbing Yu, and Mary Jean Harrold. 2009. Lightweight fault-localization using multiple coverage types. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, 56–66.

[75] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *IEEE transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[76] Johannes Schneider, Abraham Bernstein, Jan Vom Brocke, Kostadin Damevski, and David C Shepherd. 2017. Detecting plagiarism based on the creation process. *IEEE Transactions on Learning Technologies* 11, 3 (2017), 348–361.

[77] Kimberly Scott and Laura Schulz. 2017. Lookit (Part 1): A New Online Platform for Developmental Research. *Open Mind* 1, 1 (02 2017), 4–14. https://doi.org/10.1162/OPMI_a_00002

[78] William R Shadish, Thomas D Cook, and Donald T Campbell. 2002. Experimental and quasi-experimental designs for generalized causal inference. (2002).

[79] Forrest Shull, Victor Basili, Jeffrey Carver, José Carlos Maldonado, Guilherme Horta Travassos, Manoel Mendonça, and Sandra Fabbri. 2002. Replicating software engineering experiments: addressing the tacit knowledge problem. In *Proceedings international symposium on empirical software engineering*. IEEE, 7–16.

[80] Forrest Shull, Manoel G Mendoncça, Victor Basili, Jeffrey Carver, José C Maldonado, Sandra Fabbri, Guilherme Horta Travassos, and Maria Cristina Ferreira. 2004. Knowledge-sharing issues in experimental software engineering. *Empirical Software Engineering* 9, 1 (2004), 111–137.

[81] Forrest Shull, Janice Singer, and Dag IK Sjøberg. 2007. *Guide to advanced empirical software engineering*. Springer.

[82] Forrest J Shull, Jeffrey C Carver, Sira Vegas, and Natalia Juristo. 2008. The role of replications in empirical software engineering. *Empirical software engineering* 13, 2 (2008), 211–218.

[83] Janet Siegmund, Norbert Siegmund, and Sven Apel. 2015. Views on internal and external validity in empirical software engineering. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 9–19.

[84] Fábio Fagundes Silveira. 2022. Personal communication.

[85] Fábio Fagundes Silveira, Rodrigo Avancini, David de Souza França, Eduardo Martins Guerra, and Tiago Silva da Silva. 2021. Towards an Extensible Architecture for an Empirical Software Engineering Computational Platform. In *International Conference on Computational Science and Its Applications*. Springer, 231–246.

[86] Dag IK Sjøberg. 2007. Knowledge acquisition in software engineering requires sharing of data and artifacts. In *Empirical Software Engineering Issues. Critical Assessment and Future Directions*. Springer, 77–82.

[87] Dag IK Sjøberg, Bente Anda, Erik Arisholm, Tore Dyba, Magne Jorgensen, Amela Karahasanovic, Espen Frimann Koren, and Marek Vokác. 2002. Conducting realistic experiments in software engineering. In *Proceedings international symposium on empirical software engineering*. IEEE, 17–26.
[88] Dag IK Sjøberg, Bente Anda, Erik Arisholm, Tore Dybå, Magne Jørgensen, Amela Karahasanović, and Marek Vokáč. 2003. Challenges and recommendations when increasing the realism of controlled software engineering experiments. In *Empirical methods and studies in software engineering*. Springer, 24–38.
[89] Dag IK Sjøberg and Gunnar R Bergersen. 2022. Personal communication.
[90] Dag IK Sjøberg, Tore Dyba, and Magne Jorgensen. 2007. The future of empirical methods in software engineering research. In *Future of Software Engineering (FOSE'07)*. IEEE, 358–378.
[91] Dag IK Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, N-K Liborg, and Anette C Rekdal. 2005. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering* 31, 9 (2005), 733–753.
[92] Davide Spadini. 2020. CRExperiment. https://github.com/ishepard/CRExperiment. [Online; accessed 2022-03-11].
[93] Davide Spadini, Gül Çalikli, and Alberto Bacchelli. 2020. Primers or reminders? The effects of existing review comments on code review. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1171–1182.
[94] Davide Spadini, Fabio Palomba, Tobias Baum, Stefan Hanenberg, Magiel Bruntink, and Alberto Bacchelli. 2019. Test-driven code review: an empirical study. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1061–1072.
[95] Kathryn T Stolee and Sebastian Elbaum. 2010. Exploring the use of crowdsourcing to support empirical studies in software engineering. In *Proceedings of the 2010 ACM-IEEE international symposium on Empirical software engineering and measurement*. 1–4.
[96] Mohammad Tahaei and Kami Vaniea. 2022. Recruiting Participants With Programming Skills: A Comparison of Four Crowdsourcing Platforms and a CS Student Mailing List. CHI.
[97] JetBrains Team. 2015. MPS: Meta Programming System.
[98] Christopher S. Timperley, Lauren Herckis, Claire Le Goues, and Michael Hilton. 2021. Understanding and improving artifact sharing in software engineering research. *Empirical Software Engineering* 26 (2021).
[99] Koji Torii, Ken-ichi Matsumoto, Kumiyo Nakakoji, Yoshihiro Takada, Shingo Takada, and Kazuyuki Shima. 1999. Ginger2: An environment for computer-aided empirical software engineering. *IEEE Transactions on Software Engineering* 25, 4 (1999), 474–492.
[100] Guilherme H Travassos. 2022. Personal communication.
[101] Guilherme H Travassos, Paulo Sérgio Medeiros dos Santos, Paula Gomes Mian, Paula Gomes Mian Neto, and Jorge Biolchini. 2008. An environment to support large scale experimentation in software engineering. In *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2008)*. IEEE, 193–202.
[102] Sira Vegas, Natalia Juristo, Ana Moreno, Martín Solari, and Patricio Letelier. 2006. Analysis of the influence of communication between researchers on experiment replication. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. 28–37.
[103] Qianqian Wang, Chris Parnin, and Alessandro Orso. 2015. Evaluating the usefulness of ir-based fault localization techniques. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. 1–11.
[104] Gerald M Weinberg. 1971. *The Psychology of Computer Programming*. Vol. 29. Van Nostrand Reinhold New York.
[105] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: from mathematical notation to beautiful diagrams. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 144–1.
[106] YoungSeok Yoon. 2015. *Backtracking Support in Code Editing*. Ph. D. Dissertation.
[107] YoungSeok Yoon and Brad A Myers. 2011. Capturing and analyzing low-level events from the code editor. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. 25–30.
[108] YoungSeok Yoon and Brad A Myers. 2012. An exploratory study of backtracking strategies used by developers. In *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*. IEEE, 138–144.
[109] YoungSeok Yoon and Brad A. Myers. 2014. A longitudinal study of programmers' backtracking. In *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 101–108. https://doi.org/10.1109/VLHCC.2014.6883030
[110] YoungSeok Yoon and Brad A Myers. 2015. Supporting selective undo in a code editor. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 223–233.
[111] YoungSeok Yoon, Brad A Myers, and Sebon Koo. 2013. Visualization of fine-grained code change history. In *2013 IEEE symposium on visual languages and human centric computing*. IEEE, 119–126.
[112] Nico Zazworka. 2022. Personal communication.
[113] Andreas Zendler. 2001. A Preliminary Software Engineering Theory as Investigated by Published Experiments. *Empirical Software Engineering* 6, 2 (01 Jun 2001), 161–180. https://doi.org/10.1023/A:1011489321999