# Mini-Crowd Research: A Collaborative Model for Involving Beginner Researchers in Software Engineering Research

Anonymous Author(s)

## ABSTRACT

Graduate students (Master's and PhD) are crucial for the long-term sustainability of our software engineering research area. Undergraduate students are more likely to apply to graduate programs if they are engaged in research during their undergraduate studies. The traditional model to engage undergraduate students in research is via one-on-one or small-group apprenticeships with senior researchers such as professors, postdocs, or PhD students. However, the traditional model puts a heavy demand for time on senior researchers and can involve only a relatively small number of undergraduate students.

We advocate for a novel model, which we call *mini-crowd research*, to engage a larger number of undergraduate students (e.g., 10–20–30) working collaboratively on *one* research project. We build our model on the successful Crowd Research Initiative (CRI) that involved 1,500+ participants over several years. We scale down CRI in terms of the number of students and the duration to fit in a period corresponding to a summer school break, 3–4 months. This paper describes our initial experience in running a mini-crowd research project during Summer 2023.

## 1 INTRODUCTION

Research in our international software engineering community substantially depends on the efforts contributed by graduate students in Master's and PhD programs. Recruiting graduate students requires applications from undergraduate students. Research shows that undergraduate students are more likely to apply for graduate programs and to perform well in graduate programs (once accepted) if they are engaged in research projects during their undergraduate studies [6, 11, 17, 35]. As a result, including undergraduate students in our software engineering research projects can help our global community to increase the pool of graduate applicants.

Many senior researchers, such as professors, postdocs, or PhD students, from all around the world already include in research beginner researchers, such as undergraduate students or even high-school students, but primarily do so via the traditional model. In this model, a senior project lead works closely with one beginner or a small group, in an apprenticeship arrangement [18]. Such traditional model puts a heavy demand for time on the senior leads and can involve only a relatively small number of undergraduates. As a personal experience, the senior professor co-author of this paper could in the past involve only 5–6 undergrads in 2–3 projects in parallel during the school year or even during a school break.

We would love to involve many more undergraduates in *meaningful* research experiences, while not substantially increasing the demand for project lead time, especially during summer break. Our primary goal is to expose new students to the research process so that they can (1) make an informed decision whether to join industry or apply for research-based graduate programs, and (2) be better prepared for graduate programs, should they decide to pursue such programs. In particular, we *a priori* reject that the sole metric for evaluating a summer program should be how many participants were nudged/convinced to apply for graduate programs. Our secondary goal is to produce top-notch research results that could be published in the leading software engineering conferences.

While many scientific projects include crowd-sourced contributors [3, 4, 8, 26, 29, 30, 34], we are inspired by the successful Crowd Research Initiative (CRI) [32] that involved 1,500+ participants over several years in multiple projects. We refer to our model as *mini-crowd research*, because we aim for project sizes smaller than the CRI. Ideally we would prefer 10–30 undergraduates working collaboratively on *one* research project for the period corresponding to a summer school break, ~3–4 months.

This paper describes our initial offering of a mini-crowd research project during Summer 2023. We present our experience and a plan for a future similar program. In this first offering, we did not carefully track many metrics, e.g., the number of students who started but did not contribute to the project. In brief, we involved 15–20 beginner researchers who contributed to various parts of the overall project and ~10 more students who started but made no lasting contribution. We had two project leads (PIs), one postdoc and one senior professor. A key outcome of our work was a submission to a SE conference with 17 co-authors (citation omitted for double-anonymous reviewing). Side efforts also included an accepted paper (citation again omitted) and several pull requests accepted in open source. Most importantly, our project provided opportunity for beginner researchers to learn; §4 summarizes their feedback.

## 2 KEY COMPARISONS

To position our experience in the context of prior work, we compare our program to two main related lines of work. Steffen et al. [16] recently performed a software engineering study with 45 researchers. These researchers were actually performing the study, not just being subjects that were studied. Specifically, the study was on untangling commits [16]. We refer to this study as *UC*.

Vaish et al. [32] reported on the Crowd Research Initiative (*CRI*) that involved 1,500+ participants over 3+ years in multiple projects in HCI, computer vision, and data science. The results included (1) several papers with a fairly large number of co-authors (e.g., 60 authors [27], 70 authors [9], 37 authors [10], 28 authors [33], 29 authors), (2) a novel research platform Daemo [9] for crowdsourced work (not just crowd research), and (3) most importantly from the education perspective, upward mobility for participants, placing dozens of undergraduate students in graduate schools and high-school students in undergraduate schools. Of particular note is that 3/4 of the participants were from the universities ranked below 500 by Times Higher Education [2] yet their project contributions led to strong recommendation letters and acceptance in top-ranked universities in the North America and Europe [1].

We compare our approach to UC [16] and CRI [32] as follows.

- **Participants:** UC recruited only participants with formal training in computer science; in contrast, CRI and we issued an open call for all participants, but we did include selection through an evaluation task that measures some skills and motivation.
- **Ideation:** UC proposed a specific task, via a registered report [15], even before inviting the participants. The benefit is that participants can be productive from the very start, but they do not learn about the process of selecting research topics. In contrast, CRI performed a full cycle of research projects, first waiting for students to propose the specific task. However, CRI projects took longer (6+ months), while we aimed for a tangible research outcome, e.g., a submitted paper or released dataset, within a typical school break (~3–4 months). As a result, we spent some time allowing the students to propose tasks and learn by discussing these tasks, but we ended up with a task proposed by the PIs.
- **Research area:** UC and we focused on software engineering, while CRI covered HCI, computer vision, and data science.
- **Technical work:** UC tasks were on labeling a dataset, although related to code, while our tasks were on developing code. CRI had a mix of labeling tasks (e.g., in computer vision) and software development (e.g., in Daemo).
- **Collaboration:** UC assigned individual, independent tasks for labeling data. CRI had a mix of individual and collaborative tasks. In contrast, we had only collaborative code development. All approaches also had collaborative writing of research papers, but UC and CRI papers were substantially led by the project leads, while we allowed the students to take more initiative.
- **Credit:** UC introduced the notion of *research turk*, explicitly aiming "to scale the curation of manually validated data. The idea is inspired by the mechanical turk and replaces monetary payment with authorship of data set publication." [15]. The participants were evaluated based on the number of labeled data items. CRI used an elaborate scheme based on voting and network science to determine the order of paper authors. We used a more collaborative approach with team discussions to evaluate the contributions to code and ideas.
- **Communication:** UC used emails and GitHub to resolve differences in labeling, while we extensively used Discord for asynchronous communication and Zoom meetings for synchronous communication. CRI used several platforms, including Slack, YouTube Live, and "dogfooding" their own Daemo [9].
- **Learning:** UC participants could not help one another much because of the independent structure of tasks, while our participants had numerous instances where they could learn together and help one another. We even had some student-initiated and student-attended meetings, with no PI involvement. CRI participants also had extensive discussions but over a longer period of time than our project.

In sum, UC focused on *(performing) research*, whereas we focused on *(training) researchers*. CRI covered both of these key aspects.

# 3 PROGRAM ROADMAP

Our mini-crowd research program consisted broadly of 4 stages, described in the following subsections: student recruitment, ideation for research, student task allocation, and paper writing. We focus on the aspects that differ the most from the leads' prior experience (the professor and the postdoc mentored over 100 undergrads in research but not in mini-crowd) and from work such as UC [16] and CRI [32]. Fig. 1 shows the project timeline, with an estimated duration of each stage relative to the program initiation.

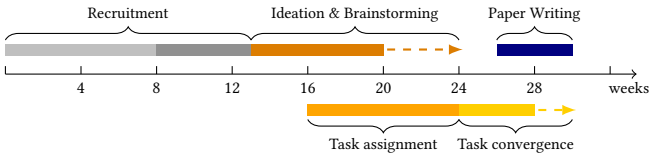



**Figure 1: Program Timeline (weeks since inception)**

## 3.1 Participant Recruitment

The recruitment started with an open "call for students" for a multi-university summer research program, advertised on social media and via personal contacts from late January through March of 2023. We explicitly stated that the research will be unpaid (unlike, say, Google Summer of Code [30]) and remote. 250+ students who filled out our online survey were emailed about our mini-crowd research experience and invited to attempt an evaluation task. The evaluation task was designed to gauge student expertise with programming and debugging, and required them to:

- Choose a bug from a dataset (details anonymized for submission)
- Verify that the bug can be reproduced (demonstrated by providing GitHub Actions for reproducing the work)
- Attempt to fix the bug (optional)

Our goal was to involve students in the comprehensive research process, from ideation to implemenation, evaluation, and paper writing. One challenge is to select an appropriate evaluation task even before the ideation stage begins. Our evaluation task could not exactly align with the project's scope before we decided what the project would be. As such, the evaluation task should evaluate students' broad skillset, ability to learn, and *motivation*. We chose a challenging task used in a homework assignment (details omitted for double-anonymous reviewing).

## 3.2 Communication Platform

Of the 250+ students who were contacted with the evaluation task, 32 qualified for the program and were emailed to vote on the online communication platform of their choice. The most voted platform was Discord, so we chose it, although the PIs were not familiar with it. While the students had themselves voted for Discord, we later got comments that it was not ideal. Some students mentioned that they did not perceive Discord as a "serious" platform for work, potentially because none of their courses used Discord for communication. We experimented with different techniques to divide discussions, using "channels" to divide high-level topics, such as discussions about tasks or meetings, and "threads" to discuss more specific sub-topics (a particular sub-task or challenge). In the future, we would likely use a platform that the PIs know better, e.g., Slack.

## 3.3 Ideation and Brainstorming

**Discussing Research Process for (Mini-)Crowds.** Given that our program had mostly undergraduates, we expected students to have little to no prior research experience. To give them context about

crowd research, they were introduced to the CRI work by Vaish et al. [32]. To start the research process, the PIs asked the participants to read this paper and answer the following questions: "What did you like about the paper? Any ideas for how the described process could be improved?". After a brief discussion, the PIs decided to proceed with a process similar as described in CRI [32].

**Self-Choosing Topics.** The next phase was to ask participants to propose specific research topics that our entire mini-crowd could work on. The participants were encouraged to read/skim recent papers from top software engineering conferences (but were *not* given specific papers to read) or to reflect on the problems they have experienced in their own software development. This phase took about three weeks and produced some small discussions on Discord and on Zoom (e.g., participants presenting slides on papers they read). However, the phase did not appear to be a productive means of identifying a viable project topic, given our timeline. In retrospect, the participants were too inexperienced in research to effectively propose a research topic. In the future, we plan to either (1) skip this phase altogether and replace it by giving students a set of topics to choose from, or (2) plan for projects longer than one summer (e.g., CRI projects typically lasted for many months).

**Lead-Proposed Topics.** To help students to narrow down a potential project to work on, the PIs proposed two specific topics related to software engineering (in a rather informal tone, characteristic of our Discord discussions):

(1) "Collecting a dataset is a kind of project that can effectively involve 30 students, some contributing manually, some automating some steps, some working more, some working less."
(2) "Translate some large-ish code from one programming language or framework to another. Again, this project can support multiple students but not as easily as collecting a dataset."

## 3.4 Lead-Proposed Tasks

Because it did not appear promising for the participants to identify a research topic doable within the summer period, the PIs assigned specific technical tasks for the participants to explore. The PIs took into account the diversity of the participants' background and assigned four tasks with different characteristics in terms of the research method (e.g., reproducing prior work vs. building a new technique) and in terms of the programming language (e.g., Python vs. Java). Specifically, the PIs proposed that the students direct their attention toward the following tasks:

(1) "There exists a bug database for Python called BugsInPy. Please select few bugs and try to reproduce them."
(2) "Evaluate the RegMiner tool on some repositories on GitHub."
(3) "Manually translate some (ideally small) microservice code to Java (or any other language not used in the original service)."
(4) "Manually translate a simple Java library to Python. You should be able to do this task even on your laptops/desktops.)"

The first three tasks had relatively low traction for a variety of reasons. Some were too difficult considering students' backgrounds. Some required more powerful computers (e.g., to run multiple Docker containers) than the students had. Some were building from research code and datasets that are not well maintained.

After some time, task 1 branched off, no one worked on task 2, one student completely translated a microservice for task 3 and another student partially translated another microservice. In the end, task 1 resulted in a short submission co-authored by only one student participant, and a few other students acknowledged for their help. Some students left or disengaged while we searched for the topic. All remaining students shifted to task 4. The PIs specifically proposed task 4 because it had a *low barrier for participation*, allowing all students to work using only their laptops and at least one language they were familiar with (Java or Python).

## 3.5 Core Technical Work

After identifying the research topic, the actual process of code translation got under way. To begin with the translation, students started with the library identified in task 4 (§3.4). The PIs first asked students to translate tests for the library, to allow contributors to get familiar with the codebase and to validate their translation of the actual code later on. To give other students some idea of how to contribute, we first had a few classes translated by the student with the most experience to be used as a reference.

To minimize inter-dependencies between contributors, we used a reverse topological sort to map out the class dependencies in the project under translation. Once the graph was generated, students could mark a class as "In Progress" and begin their work. To reduce work duplication, students tracked their progress on GitHub issues visible to all participants. We also had some internal discussion to use a dedicated management tool (such as Trello), but the PIs considered them unnecessary for our mini-crowd scale. However, some students faced challenges in the work distribution and reported ambiguity in task assignment (see §4.4 for survey results).

To store our work, we used a common repository (monorepo) on GitHub for all related artifacts One advantage of having a monorepo was access to all available work done, with a high degree of visibility on the progress of the translation. Pros and cons of monorepos are widely reported in literature [12, 19, 25]. We had some incidents where students put their work in the wrong place, or directly committed incorrect code, leading to extensive untangling of commits to revert any breaking changes.

As a result, such incidents encouraged us to add Continouous Integration (CI) workflows to our repository, to run tests before code was merged in the main tree. We added our first CI workflow 4 weeks into the "Task Assignment" stage (Fig. 1) that carried out selective regression testing, and only ran on parts of the project that were modified in the pull request (PR). We also added workflows for style checks for both languages (Python and Java) highlight any inconsistencies at review time. All PRs required one or more approvals from collaborators (PIs or code owners) before being merged into the main branch of the tree.

Other than work on code translation, students in the program were also able to draft an evaluation task that directly related to the current project. This task can help recruit more students as contributors, and scale our program to continue work in the future.

Fig. 2 aggregates the number of pull requests, issues, comments, and reviews over time, showing the initial lag in levels of contributions but the gradual rise as students overcame the steep learning curve (§ 4.2 has more details). Each line represents one participant.

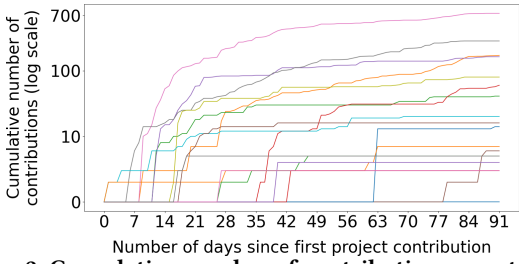
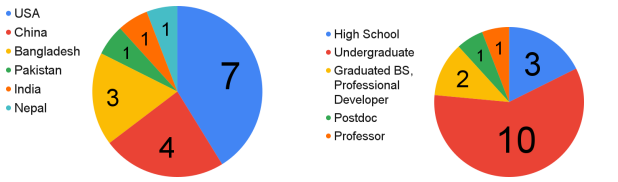



**Figure 2: Cumulative number of contributions per student**




**(a) Distribution by country** **(b) Distribution by academic level**

**Figure 3: Demographics of our mini-crowd participants**

## 3.6 Timeline

Fig. 1 shows the timeline for the mini-crowd research, with an approximate duration for each stage. After half of the program, we got enough work done to aim for a submission to an SE conference. The leads announced on Discord that the entire team may submit a paper if we make significant progress in our code translation. We also announced that the submission co-authors will be only those participants who have made or will make *non-trivial contributions* to the project before the submission deadline. Providing the specific deadline and motivation to co-author a paper substantially increased the progress on the project, especially as we got closer to the SE conference deadline. For a future mini-crowd research program, we would evaluate how different methods of mentorship affect student's commitment levels, as this could create a greater long-term impact in the overall research experience [21].

## 3.7 Convergence: Writing a paper

Once it became clear that we can aim for a submission, participants were asked to compile specific challenges that they faced within a shared document. This document eventually grew into a repository of data which included bugs, challenges faced during translation, performance analyses, and statistical reports. In the days near the submission, we had dedicated Zoom meetings to discuss progress and challenges with the most active contributors, as there was increased emphasis on finishing the translation in time.

The paper itself was written by one of the PIs, along with 2–3 of the most active students, while other students only provided smaller comments or feedback on the text. We were able to successfully submit a paper to a SE conference (details omitted for double-anonymous review) with a total of 17 co-authors.

**Co-author demographics:** Fig. 3 shows the geographic distribution and the academic qualifications of the 17 co-authors. They came from a total of 6 countries, including 10 undergraduate students, 3 high-school students, and 2 professional software developers.

# 4 PARTICIPANT SURVEY

After submitting to an SE conference, we conducted a survey of all participants to gather their feedback on the mini-crowd research experience. We asked the following questions:

(1) What skills did you learn during the summer research?
(2) What were some personal/technical challenges that you faced?
(3) What aspects of the mini-crowd research did you like?
(4) What aspects of the mini-crowd research could be better?

We group students' responses to questions 1, 2, and 3 according to common categories and count the frequency of each category. Question 4 is the most important for future iterations, and to expose students to qualitative research [7], we follow a more systematic coding approach [31]: We categorize the answers and determine sentiments from student responses. For question 4, two co-authors collectively analyzed the responses and agreed on a final coding.

## 4.1 Perceived Learning

**Technical Skills (29).** The most common perceived learning category was technical skills, where students reported learning about the technical content of our paper submission and the process needed to reach that stage. One student noted: "[I] learned software development skills, usage of Python, and other external tools."
**Collaboration and Communication (13).** The second most common category was students learning how to contribute to a larger project by communicating their ideas and progress, and by collaborating effectively.
**Research and Academia (7).** The third most common category referred to students' abilities to hone their skills in understanding research papers and to gain insights into research methods. One student reported achieving a better balance of "optimizing understanding vs time spent reading" research papers. The PIs assume that all students learned important and transferable research skills [4] but did not report them when asked about general "skills".
**Documentation and Presentation (2).** The least common reported category was students learning how to document their progress (e.g., in pull requests on GitHub) and share their findings during Zoom meetings. Yet again, the PIs believe that the students greatly improved but are not even aware of their improvements.

## 4.2 Challenges Encountered

**Technical Challenges (11).** The most commonly reported challenges were technical, including adapting to using GitHub, working with the compiler framework (details omitted for double-blind review), or creating paper artifacts.
**Personal Challenges (9).** Many students faced personal challenges such as adjusting to Zoom meeting times (with participants in timezones up to 12 hours apart) and health-related issues.
**Steep Learning Curve (7).** The third most common category was the steep learning curve, referring to the complexity of the program and the need for students to acquire new knowledge rapidly in the starting phase. One student noted "getting up to speed on the technologies and collaboration tools involved" was challenging.
**Technological (3).** The least common category included the need for a VPN to use Discord in China and the setup of necessary software and configurations on a student's computer.

### 4.3 Positive Aspects

**Positive Work Environment (13).** This category was the most commonly reported positive of our research program. Students received substantial support, especially for beginners to software engineering. Participants praised others as accountable, collaborative, and cooperative, e.g., a student described "working with a number of excellent and friendly people is so wonderful, which is still inspiring and motivating me now." We are particularly happy for students who would not have had other opportunities to be engaged in research due to their country or university.

**Remote Work and Flexibility (10).** The remote nature of the program and the flexibility it offered participants in terms of time commitments and input levels was also a great positive. The leads emphasized that the students can put in as much or as little time as they want, but the contributions will be reflected in the author order on potential submissions and in the recommendation letters.

**Nature of Research (5).** The participants were exposed to the numerous avenues and opportunities for exploration in the mini-crowd research experience, under the guidance from the PIs.

**Technical Aspects (3).** The surprisingly infrequently reported positive was technical aspects, including some student's appreciation for the evaluation task and the use of GitHub to organize the code.

**Community Engagement (1).** Finally, the mini-crowd research fostered community engagement and interest in software engineering projects, and students were motivated when one student published a related paper while just joining our mini-crowd team.

### 4.4 Potential Improvements

**Lack of Documentation for Newcomers (6).** The most commonly reported negative was the lack of documentation or onboarding for newcomers. One student noted that detailed documentation "would clarify expectations for tasks such as pull requests, reducing the need for repetitive feedback and expediting progress".

**Communication Challenges (5).** The second most commonly reported negative category, including the lack of progress updates through Discord as an example issue faced by the student.

**Inefficient/Infrequent Meetings (5).** Several students reported that meetings did not allow the students to focus on one task, and having multiple tasks diluted the quality of the meetings.

**Ineffective Task Tracking (4).** This feedback was more common during the project than in the survey. One student reported the task tracking and distribution being unorganized, "some tracked through GitHub issues, some within Discord and some via Excel".

**Unclear goals/focus (3).** The least reported category, but likely the most important, was that some students found that for much of the duration of the project they were unsure of the ultimate objective, which reduced participation and motivation levels.

### 4.5 Non-Survey Feedback

Several students left our mini-crowd project, and we had no "exit interviews" [14]. Most students just quietly disenaged, and some cited lack of time before leaving. One student provided a particularly honest and insightful feedback: *"I don't feel like continuing with this project anymore. Reasons for the same: 1. Lack of clarity in terms of the final goal that we are trying to achieve with this. 2. There seems to be an over-emphasis on publishing a paper without actually reasoning whether the paper would create an impact later in the industry. 3. Due to the above reasons, I was never motivated to work on the project as trying to achieve a goal, rather it just became a way of improving software-engineering skills.*
*Yes, I agree that there were many 'goods' in the project too but I thought of highlighting these points to you as I have always seen you being open to taking feedback for improving your future programs."*

## 5 RELATED WORK

Our mini-crowd research program is not the first model to involve beginner researchers (at scale). §2 already described our main influences. We review some of the most recent work.

Li et al. [20] and Nguyen et al. [24] study and evaluate programs relating to scaling research collaborations at different levels. In contrast to their strict schedules, our program had no roadmap, and students had flexibility to choose what they worked on.

Sharma et al. [28] share insights into common obstacles hindering undergraduate research experiences. Many of the barriers they identified arose in our project as participants reported their struggle with balancing other time commitments or were confused about the ultimate project goals.

Many research papers evaluated different methods of research recruitment and organization. Lykourentzou et al. [22] propose Self-Organizing Pairs for online collaboration among individuals with no prior collaborative experience. In our mini-crowd program, students frequently self-organized into smaller groups, often pairs, to address specific project aspects, which allowed efficient knowledge transfer and completion of several tasks simultaneously.

Arony et al. [23] and Graßl et al. [13] analyze conditions related to diverse student teams, studying the impact of team diversity on stress and learning outcomes, and emphasizing the balance between autonomy and guidance, and the development of technical and soft skills. Our program has similarities as our student population was diverse, and many students had similar opinions about perceived learning and challenges, and the independence vs. guidance (§4).

Dalpiaz [5] describes crowd-based requirements engineering as having potential value in both practice and research. The author's reflections on his own experience with CrowdRE in practice similarly mirrors the nature of our project including the importance of establishing a clear research goal and active moderation by leaders.

## 6 CONCLUSIONS

Mentoring undergraduate reseachers requires significant time and effort, and is traditionally done only for small groups. To democratize this opportunity for more students, we propose a *mini-crowd* research model where a team of 10–30 students contributes to a single research goal. Our first mini-crowd experience started with 32 students. We received some contributions from 15–20 students who remained with us for the program duration. We allowed students to experience most of the research process, including ideation. Our mini-crowd approach enabled 15 students to effectively contribute to one project, co-authoring a paper submitted to SE conference. We hope that our approach will motivate others to experiment with similar research programs for larger groups of undergraduates.

**Data Availability:** We could release only survey responses but after carefully redacting them to not reveal the participant identity.

# REFERENCES


[1] 2017. Crowd Research Initiative. http://crowdresearch.stanford.edu/initiative.

[2] 2023. The Times Higher Education World University Rankings. https://timeshighereducation.com/world-university-rankings/2023/world-ranking.

[3] H Bassi, L Misener, and AM Johnson. 2020. Crowdsourcing for research: perspectives from a Delphi panel. *SAGE Open* 10, 4 (2020).

[4] Tom Bourner, Linda Heath, and Asher Rospigliosi. 2014. Research as a transferable skill in higher education. *Higher education review* 46, 2 (2014), 20–46.

[5] Fabiano Dalpiaz. 2021. On the Value of CrowdRE in Research and Practice. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*.

[6] M Kevin Eagan Jr, Sylvia Hurtado, Mitchell J Chang, Gina A Garcia, Felisha A Herrera, and Juan C Garibay. 2013. Making a Difference in Science Education: The Impact of Undergraduate Research Programs. *American Educational Research Journal* (2013).

[7] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. Selecting empirical methods for software engineering research. *Guide to advanced empirical software engineering* (2008), 285–311.

[8] Lina Eklund, Isabell Stamm, and Wanda Katja Liebermann. 2019. The crowd in crowdsourcing: Crowdsourcing as a pragmatic research method. *First Monday* 4, 10 (2019).

[9] Snehal Gaikwad, Durim Morina, Rohit Nistala, Megha Agarwal, Alison Cossette, Radhika Bhanu, Saiph Savage, Vishwajeet Narwal, Karan Rajpal, Jeff Regino, et al. 2015. Daemo: A self-governed crowdsourcing marketplace. In *Adjunct proceedings of the 28th annual ACM symposium on user interface software & technology*.

[10] Snehalkumar (Neil) S. Gaikwad, Durim Morina, Adam Ginzberg, Catherine Mullings, Shirish Goyal, Dilrukshi Gamage, Christopher Diemert, Mathias Burton, Sharon Zhou, Mark Whiting, Karolina Ziulkoski, Alipta Ballav, Aaron Gilbee, Senadhipathige S. Niranga, Vibhor Sehgal, Jasmine Lin, Leonardy Kristianto, Angela Richmond-Fuller, Jeff Regino, Nalin Chhibber, Dinesh Majeti, Sachin Sharma, Kamila Mananova, Dinesh Dhakal, William Dai, Victoria Purynova, Samarth Sandeep, Varshine Chandrakanthan, Tejas Sarma, Sekandar Matin, Ahmed Nasser, Rohit Nistala, Alexander Stolzoff, Kristy Milland, Vinayak Mathur, Rajan Vaish, and Michael S. Bernstein. 2016. Boomerang. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. https://doi.org/10.1145/2984511.2984542

[11] Joanna Gilmore, Michelle Vieyra, Briana Timmerman, David Feldon, and Michelle Maher. 2015. The relationship between undergraduate research participation and subsequent research performance of early career STEM graduate students. *The Journal of Higher Education* 86, 6 (2015), 834–863.

[12] Marco Glorie, Andy Zaidman, Arie van Deursen, and Lennart Hofland. 2009. Splitting a large software repository for easing future software evolution—an industrial experience report. *Journal of Software Maintenance and Evolution: Research and Practice* 21, 2 (2009), 113–141.

[13] Isabella Graßl, Gordon Fraser, Stefan Trieflinger, and Marco Kuhrmann. 2023. Exposing Software Engineering Students to Stressful Projects: Does Diversity Matter? (2023).

[14] Don H Harris. 2000. The Benefits of Exit Interviews. *IEEE Engineering Management Review* 28, 3 (2000), 63–66.

[15] Steffen Herbold. 2020. With Registered Reports Towards Large Scale Data Curation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*.

[16] Steffen Herbold, Alexander Trautsch, Benjamin Ledel, Alireza Aghamohammadi, Taher A. Ghaleb, Kuljit Kaur Chahal, Tim Bossenmaier, Bhaveet Nagaria, Philip Makedonski, Matin Nili Ahmadabadi, Kristof Szabados, Helge Spieker, Matej Madeja, Nathaniel Hoy, Valentina Lenarduzzi, Shangwen Wang, Gema Rodríguez-Pérez, Ricardo Colomo-Palacios, Roberto Verdecchia, Paramvir Singh, Yihao Qin, Debasish Chakroborti, Willard Davis, Vijay Walunj, Hongjun Wu, Diego Marcilio, Omar Alam, Abdullah Aldaeej, Idan Amit, Burak Turhan, Simon Eismann, Anna-Katharina Wickert, Ivano Malavolta, Matúš Sulír, Fatemeh Fard, Austin Z. Henley, Stratos Kourtzanidis, Eray Tuzun, Christoph Treude, Simin Maleki Shamasbi, Ivan Pashchenko, Marvin Wyrich, James Davis, Alexander Serebrenik, Ella Albrecht, Ethem Utku Aktas, Daniel Strüber, and Johannes Erbel. 2022. A Fine-grained Data Set and Analysis of Tangling in Bug Fixing Commits. *Empirical Software Engineering* 27, 6 (2022), 125.

[17] Paul R Hernandez, Anna Woodcock, Mica Estrada, and P Wesley Schultz. 2018. Undergraduate research experiences broaden diversity in the scientific workforce. *BioScience* 68, 3 (2018), 204–211.

[18] Anne-Barrie Hunter, Sandra L Laursen, and Elaine Seymour. 2007. Becoming a scientist: The role of undergraduate research in students' cognitive, personal, and professional development. *Science Education* 91 (2007), 36–74.

[19] Ciera Jaspan, Matthew Jorde, Andrea Knight, Caitlin Sadowski, Edward K. Smith, Collin Winter, and Emerson Murphy-Hill. 2018. Advantages and Disadvantages of a Monolithic Repository: A Case Study at Google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. https://doi.org/10.1145/3183519.3183550

[20] Ze Shi Li, Nowshin Nawar Arony, Kezia Devathasan, and Daniela Damian. 2023. "Software is the easy part of Software Engineering" - Lessons and Experiences from A Large-Scale, Multi-Team Capstone Course. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. https://doi.org/10.1109/ICSE-SEET58685.2023.00027

[21] Marcia C Linn, Erin Palmer, Anne Baranger, Elizabeth Gerard, and Elisa Stone. 2015. Undergraduate research experiences: Impacts and opportunities. *Science* 347, 6222 (2015), 1261757.

[22] Ioanna Lykourentzou, Federica Lucia Vinella, Faez Ahmed, Costas Papastathis, Konstantinos Papangelis, Vassilis-Javed Khan, and Judith Masthoff. 2022. Self-organization in online collaborative work settings. *CoRR* (2022).

[23] Nowshin Nawar Arony, Kezia Devathasan, Ze Shi Li, and Daniela Damian. 2023. Leveraging Diversity in Software Engineering Education through Community Engaged Learning and a Supportive Network. (2023), 247–258. https://doi.org/10.1109/ICSE-SEET58685.2023.00029

[24] David Van Nguyen, Daniel A Epstein, and Shayan Doroudi. 2023. Effects of Scaling Up Apprentice-Style Research: Perceptions from Mentors and Mentees. In *Proceedings of the Tenth ACM Conference on Learning@ Scale*.

[25] Rachel Potvin and Josh Levenberg. 2016. Why Google Stores Billions of Lines of Code in a Single Repository. *Commun. ACM* (2016).

[26] Kaja Scheliga, Sascha Friesike, Cornelius Puschmann, and Benedikt Fecher. 2018. Setting up crowd science projects. *Public Understanding of Science* 27, 5 (2018), 515–534.

[27] Alok Shankar Mysore, Vikas S. Yaligar, Imanol Arrieta Ibarra, Camelia Simoiu, Sharad Goel, Ramesh Arvind, Chiraag Sumanth, Arvind Srikantan, Bhargav HS, Mayank Pahadia, Tushar Dobha, Atif Ahmed, Mani Shankar, Himani Agarwal, Rajat Agarwal, Sai Anirudh-Kondaveeti, Shashank Arun-Gokhale, Aayush Attri, Arpita Chandra, Yogitha Chilukur, Sharath Dharmaji, Deepak Garg, Naman Gupta, Paras Gupta, Glincy Mary Jacob, Siddharth Jain, Shashank Joshi, Tarun Khajuria, Sameeksha Khillan, Sandeep Konam, Praveen Kumar-Kolla, Sahil Loomba, Rachit Madan, Akshansh Maharaja, Vidit Mathur, Bharat Munshi, Mohammed Nawazish, Venkata Neehar-Kurukunda, Venkat Nirmal-Gavarraju, Sonali Parashar, Harsh Parikh, Avinash Paritala, Amit Patil, Rahul Phatak, Mandar Pradhan, Abhilasha Ravichander, Krishna Sangeeth, Sreecharan Sankaranarayanan, Vibhor Sehgal, Ashrith Sheshan, Suprajha Shibiraj, Aditya Singh, Anjali Singh, Prashant Sinha, Pushkin Soni, Bipin Thomas, Kasyap Varma-Dattada, Sukanya Venkataraman, Pulkit Verma, and Ishan Yelurwar. 2015. Investigating the "Wisdom of Crowds" at Scale. In *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. https://doi.org/10.1145/2815585.2815725

[28] Rhea Sharma, Atira Nair, Ana Guo, Dustin Palea, and David T Lee. 2022. "It's usually not worth the effort unless you get really lucky": Barriers to Undergraduate Research Experiences from the Perspective of Computing Faculty. In *ICER '22*. Association for Computing Machinery.

[29] Raphael Silberzahn and Eric L Uhlmann. 2015. Crowdsourced research: Many hands make tight work. *Nature* 526, 7572 (2015), 189–191.

[30] Jefferson O Silva, Igor Wiese, Daniel M German, Christoph Treude, Marco A Gerosa, and Igor Steinmacher. 2020. Google summer of code: Student motivations and contributions. *Journal of Systems and Software* 162 (2020), 110487.

[31] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *ICSE*.

[32] Rajan Vaish, Snehalkumar (Neil) S. Gaikwad, Geza Kovacs, Andreas Veit, Ranjay Krishna, Imanol Arrieta Ibarra, Camelia Simoiu, Michael Wilber, Serge Belongie, Sharad Goel, James Davis, and Michael S. Bernstein. 2017. Crowd Research: Open and Scalable University Laboratories. In *UIST*. https://doi.org/10.1145/3126594.3126648

[33] Mark E. Whiting, Dilrukshi Gamage, Snehalkumar (Neil) S. Gaikwad, Aaron Gilbee, Shirish Goyal, Alipta Ballav, Dinesh Majeti, Nalin Chhibber, Angela Richmond-Fuller, Freddie Vargus, Tejas Seshadri Sarma, Varshine Chandrakanthan, Teogenes Moura, Mohamed Hashim Salih, Gabriel Bayomi Tinoco Kalejaiye, Adam Ginzberg, Catherine A. Mullings, Yoni Dayan, Kristy Milland, Henrique Orefice, Jeff Regino, Sayna Parsi, Kunz Mainali, Vibhor Sehgal, Sekandar Matin, Akshansh Sinha, Rajan Vaish, and Michael S. Bernstein. 2017. Crowd Guilds. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. https://doi.org/10.1145/2998181.2998234

[34] Yuxiang Zhao and Qinghua Zhu. 2014. Evaluation on crowdsourcing research: Current status and future direction. *Information systems frontiers* 16 (2014), 417–434.

[35] Andrew L Zydney, Joan S Bennett, Abdus Shahid, and Karen W Bauer. 2002. Impact of Undergraduate Research Experience in Engineering. *Journal of Engineering Education* (2002).