

# **On-Demand Collaboration in Programming**

by

Yan Chen

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Information)  
in the University of Michigan  
2020

Doctoral Committee:

Professor Steve Oney, Chair  
Professor Mark Ackerman  
Professor Mark Guzdial  
Professor Philip Guo

Yan Chen

yanchenm@umich.edu

ORCID iD: 0000-0002-1646-6935

©Yan Chen 2020

# Acknowledgments

I am very grateful that I have been given the opportunity over the past six years to conduct my research projects and turn them into my Ph.D. dissertation. This unforgettable journey would not have been possible without the support of many people and organizations.

I want to thank Steve Oney, my Ph.D. advisor, for his mentorship throughout my Ph.D. Steve was always available to provide expert support to my on-demand research questions. This support was very personalized and customized, and has always guided me through the “dark road.” I also want to thank Mark Ackerman, Mark Guzdial, and Philip Guo for serving my dissertation committee and provide constructive and critical feedback from different perspective. I also want to thank Walter Lasecki, my previous Ph.D. advisor, who has always been pushing me to think deeper about the research problems.

I would also like to thank my internship mentors Andrés Monroy-Hernández, Rajan Vaish, and Ian Wehrman, at Snapchat, and Tao Dong at Google. They opened my views on research process, topics, and many other perspectives. I also want to thank the colleagues who directly helped me on the research projects that comprise this dissertation: April Wang, Divya Ramesh, Gabriel Matute, Harmanpreet Kaur, Jasmine Jones, Jaylin Herskovitz, Jean Y. Song, John Chung, Jonathan Kummerfeld, Jordan Huffaker, Lei Zhang, Mauli Pandey, Sai R. Gouravajhala, Sang Won Lee, Stephanie O’Keefe, Yaxing Yao, and Yiwei Yang. Lastly, I would like to thank my girlfriend Rebecca Krosnick, and my parents, Ziyang Xie and Jun Chen for their ongoing support.

# Table of Contents

<b>Acknowledgments</b>	<b>ii</b>
<b>List of Tables</b>	<b>viii</b>
<b>List of Figures</b>	<b>x</b>
<b>Abstract</b>	<b>xiv</b>

## Chapters

<b>1 Introduction</b>	<b>1</b>
1.1 On-demand Collaboration With Expert Crowds . . . . .	2
1.2 Two Need-finding Studies . . . . .	5
1.3 Designing On-Demand Programming Assistance . . . . .	5
1.4 CoCapture: Effectively Communicating UI Behaviors on Existing Websites by Demonstrating and Remixing . . . . .	6
1.5 Supporting Remote Collaboration for Embedded System Development . . . . .	7
1.6 Thesis Statement and Contributions . . . . .	8
1.7 Thesis Outline . . . . .	9
<b>2 Related Work</b>	<b>10</b>
2.1 Collaboration in Software Development . . . . .	10
2.1.1 Community Question Answering . . . . .	10
2.1.2 IDE-Integrated Help Finding Tools . . . . .	11
2.1.3 Programming Team Communication . . . . .	11
2.1.4 Pair Programming . . . . .	12
2.1.5 Information Needs for Developers . . . . .	12
2.1.6 Collaborative Development . . . . .	12
2.2 Human Computation in Software Development . . . . .	13



2.2.1	Real-Time Crowdsourcing	13
2.2.2	Crowd-Enabled IDE Tools	14
2.2.3	Expert Guidance	14
2.2.4	Expert crowds	15
2.3	Communicating UI Behaviors	16
2.3.1	Creating UI Prototypes and Mockups	16
2.3.2	Visual References as Shared Context in Communication	17
2.3.3	Record, Replay, and Manipulate Existing Interfaces	18
2.4	Remote Collaboration for Embedded System Development	19
2.4.1	Remote Collaboration During Physical Tasks via Audio/Video Techniques	19
2.4.2	Support for Remote Collaborative Tasks via Augmented Reality Techniques	20
2.4.3	Remote Collaboration for Embedded System Development	20
2.4.4	Teleoperated Robots for Remote Collaboration	21
<b>3</b>	<b>On-Demand Expert Support Paradigm</b>	<b>23</b>
3.1	Motivating Scenario	23
3.1.1	Comments and Discussion	24
3.2	Challenges	24
3.3	Study 1 Design	25
3.4	Participants' Request Categories	26
3.4.1	Memory Aids	26
3.4.2	Explanatory Requests	26
3.4.3	High-Level Strategic Guidance	26
3.4.4	Code Requests	27
3.4.5	Debugging Requests	27
3.4.6	Code Refactoring	28
3.4.7	Effort-Saving Requests	28
3.5	Findings and Interviews	28
3.6	Study 2 Design	29
3.7	Findings	31
3.7.1	Experience	32
3.7.2	Context	33

3.7.3	Sharing . . . . .	33
3.7.4	Real-time Response . . . . .	34
3.7.5	Integrated System . . . . .	34
3.7.6	Personalized Help . . . . .	35
3.8	Discussions and System Design Suggestions . . . . .	35
3.8.1	Helper Page . . . . .	35
3.8.2	Suggested Support Features . . . . .	36
3.9	Chapter Conclusion . . . . .	39
<b>4</b>	<b>CodeOn: On-Demand Programming Assistance</b>	<b>40</b>
4.1	Introduction . . . . .	40
4.2	Three studies to motivate CodeOn’s design . . . . .	40
4.2.1	Stage 1: Making a Request . . . . .	42
4.2.2	Stage 2: Writing a response . . . . .	43
4.2.3	Stage 3: Integrating a response . . . . .	45
4.3	Iterative Design of the End-To-End System . . . . .	48
4.4	Experiment . . . . .	48
4.5	Hypotheses . . . . .	49
4.6	Overall Performance . . . . .	49
4.7	Individual Task Performance . . . . .	50
4.8	Interruptions and Parallelization . . . . .	52
4.9	Interview and Developer Feedback . . . . .	54
4.9.1	Parallelization . . . . .	54
4.9.2	Interruption . . . . .	54
4.10	Discussion . . . . .	55
4.10.1	CodeOn User Interface . . . . .	55
4.10.2	Effects of Social Norms . . . . .	56
4.10.3	Potential of CodeOn . . . . .	56
4.10.4	Generalizing CodeOn’s Approach . . . . .	57
4.11	Chapter Conclusion . . . . .	57
<b>5</b>	<b>CoCapture: Facilitating Communication About Dynamic Interactive UI Behav-</b>	
	<b>iors</b>	<b>58</b>

5.1	Introduction . . . . .	58
5.2	Need-finding Studies . . . . .	60
5.2.1	Stack Overflow Analysis . . . . .	60
5.2.2	Needs and Challenges . . . . .	61
5.2.3	Design Goals . . . . .	63
5.3	CoCapture Design and Implementation . . . . .	63
5.3.1	The CoCapture User Experience . . . . .	64
5.3.2	Design and Implementation . . . . .	67
5.4	System Evaluation . . . . .	72
5.4.1	Study 1 - Creating UI Behavior Questions . . . . .	72
5.4.2	Results and Analysis . . . . .	73
5.4.3	System Usability and Study Insights . . . . .	75
5.5	Study 2 - Understanding UI Behavior Questions . . . . .	77
5.5.1	CoCapture Showed Potential in Facilitating Effective Comprehension of User Interface (UI) Questions . . . . .	78
5.5.2	CoCapture Questions Were Clearly Presented . . . . .	78
5.6	Discussion . . . . .	79
5.6.1	Element-based Animations as First-class Objects . . . . .	79
5.6.2	Hypertext Helps Organize and Ground Communication . . . . .	80
5.6.3	System Scope and Limitations . . . . .	81
5.6.4	Future Work . . . . .	81
5.7	Chapter Conclusion . . . . .	82
<b>6</b>	<b>Supporting Remote Collaboration for Embedded System Development</b>	<b>83</b>
6.1	Introduction . . . . .	83
6.2	Study Design . . . . .	85
6.2.1	Semi-structured Interview Protocol . . . . .	85
6.2.2	Hypothetical Assistant Study Protocol . . . . .	85
6.2.3	Participants . . . . .	89
6.3	Findings . . . . .	89
6.3.1	Practices and Challenges of On-Demand Remote Collaboration in Em- bedded System Development . . . . .	90
6.3.2	Participants' request categories for the hypothetical assistant . . . . .	94

6.3.3	Findings and Follow Up Interviews . . . . .	96
6.4	Discussion . . . . .	99
6.4.1	Providing“Beyond Being There” Technology . . . . .	99
6.4.2	Adding Automation for High Precision and Easy Control for the Robot	101
6.4.3	Specialized Robot Arm for Embedded System Development Tasks . . .	102
6.4.4	Identity, trust, expertise sharing, and capability . . . . .	103
6.5	Chapter Conclusion . . . . .	103
<b>7</b>	<b>Conclusion</b>	<b>104</b>
7.1	Recap of Each Project . . . . .	104
7.2	Contributions . . . . .	105
7.2.1	Summary of Contributions . . . . .	107
7.3	Novelty of the Work . . . . .	108
7.4	Benefits Beyond Software Development . . . . .	109
7.5	The Future of Work . . . . .	111
	<b>Bibliography</b>	<b>114</b>

# List of Tables

4.1	Time to make a request per condition (avg./s.d.). We found that spoken requests (“Voice”) where developers highlighted the relevant context were the fastest for developers to specify. . . . .	41
4.2	The average time (in minutes) spent per task. Participants spent longer in the control condition, on both completed and incomplete tasks. Tasks in tail condition is the task that are stopped by the researchers by the time constraints (30 min). Note that, by definition, there cannot be complete task in the tail condition. . . . .	50
4.3	The # of requests made per completed task is not significantly different. The average system active time per completed task in CodeOn is longer than in the control condition (avg./s.d.). . . . .	51
4.4	# of interruptions, alerts, and average of individual ratio of interruption/alert(Avg. / S.D.). . . . .	52
4.5	Time spent and the number of parallelization behavior in two conditions (Average / S.D.). . . . .	54
5.1	Measurements from 15 participants using CoCapture and control tools (email + Google Drawings). Description accuracy (%) is calculated using (the number of satisfied items in a rubric/ total number of items). Visual references include images, sketches, and animations. ** indicates $p < .001$ , *** indicates $p < .0001$ . Their corresponding standard deviations are in parenthesis. . . . .	74

6.1	Participants’ background information. For the Participant ID (PID) column, P1-P11 represent the 11 embedded system developers. For the Gender column, M = male and F = female. For the occupation column, P = professional developers (paid to develop embedded system projects) and S = student developers. Programming skill, electronics skill, and collaboration experience are on a 7-point Likert scale where 7 means expert and 1 means no experience. . . . .	89
6.2	Common query types observed during our hypothetical assistant study, with corresponding frequencies. Each of these query types suggests a support role that remote embedded system development assistants can play in future systems. “# Sessions” indicates the number of different sessions that each type of question occurred in, while “# / Session” indicates the number of queries that referred to solving one of these query types, on average. . . . .	94
6.3	Participants’ project descriptions and progress for their hypothetical assistant study. P8’s project contains private information. . . . .	97

# List of Figures

1.1	Diagram of on-demand expert programming support paradigm. . . . .	2
1.2	Asynchronous interactions between developers and helpers can occur in three stages: making a request, writing a response, and integrating the response. . . .	6
3.1	Setup for our hypothetical assistant study. Developer participants were asked to bring their own task to complete, and ask questions from our hypothetical assistant as if it could answer any support question needed. Participants were still allowed to use traditional online resources and augment them with the assistant as they saw fit. Context related to their programming task was collected at the beginning of each study, and audio of their questions was recorded during the session. . . . .	25
3.2	Common query types observed during our hypothetical assistant study, with corresponding frequencies. Each of these query types suggests a support role that remote software development assistants can play in future systems. “Number of Sessions” indicates the number of different sessions that each type of question occurred in, while “Number of Queries per Session” indicates the number of queries that referred to solving one of these query types, on average. . . . .	27
3.3	Setup for our human expert assistant study. In each trial, one “requester” participant (developer) was paired with one “helper” (expert, based on a pre-study skill assessment). Participants could chat via either text or voice, and requesters were able to share their screen with helpers as needed using Skype. The helper was tasked with assisting the requester <i>reactively</i> , meaning that they only responded to queries, and did not proactively propose solutions or approaches. This simulates a “best case” (repeated, non-multiplexed helper) on-demand model where human experts are not expected to be continuously available between end-user queries. . . . .	31

3.4	Common participant information needs that we observed during our human expert assistant study, with corresponding frequencies. Each of these needs would limit the success of a naive approach to providing remote assistance for software developers. “Number of Sessions” indicates the number of different trial sessions that each information need occurred in, while “Number of Conversations per Session” indicates the average number of times each need arose in a conversation (stemming from requester queries). “Number of Interviews” indicates the number of unique interviews in which the need was mentioned at least once. “Number of Mentions per Interview” is the average number of times that a participant mentioned the need in the post-trial interviews. . . . .	32
4.1	Asynchronous interactions between developers and helpers can occur in three stages: making a request (S1), writing a response (S2), and integrating the response (S3). In CodeOn, developers use an IDE plug-in to make requests (S1) and integrate responses (S3), and helpers use a web-based IDE to view content and generate responses (S2). . . . .	41
4.2	CodeOn interface where the requests and responses are on the right panel(2). Developer’s code(1) and helper’s code(3) are side by side for better comparison. Other responses includes explanation(4), annotation(5), and comments(6) . . .	42
4.3	Advantages, disadvantages, and design takeaways for three response formats from developers and helpers’ perspectives). . . . .	44
4.4	The helper side of CodeOn is an interactive webpage that allows helpers to replay the request(0) and run the code(4). Helpers can respond to it with explanation(1), and inline code(2), and annotation(3). . . . .	46
4.5	Overall result: The # of completed tasks is significantly more in CodeOn condition(avg./s.d.). . . . .	50



5.1	The Workflow of CoCapture. There are four steps of using CoCapture to communicate new UI behavior mockups on an existing website. <b>(Step 1)</b> Users first capture existing interface behaviors (base scene) by interacting with the website (scrolling), and CoCapture will automatically capture the Document Object Model (DOM) changes. <b>(Step 2)</b> In CoCapture’s main panel, users can add new behaviors on top of the base scene by demonstration; that is, by directly manipulating any elements (e.g., drag and drop the red element in the replay and see immediate changes). <b>(Step 3)</b> Users can remix (post-edit, e.g., change duration) added behaviors to finalize the mockup. <b>(Step 4)</b> . Users can refer to the DOM elements or added behaviors in the textual description using hypertext.	59
5.2	CoCapture’s main panel (after Step 1 in Fig. 5.1). To create an animation, a user first clicks an existing DOM element (b1) (and can select multiple by holding the Shift key) from the base scene (B). Once recording is started (A), the user can demonstrate the desired behaviors by changing the elements’ properties (C) or directly dragging them in the base scene. Once the recording is finished, the demonstration will be appended to the animation list (D) with a set of meta operations including options to adjust the starting/ending time, replay, and preview. The user can write their question (E) and refer to the animations or DOM elements in the scene by selecting portions of text and clicking the “Link to a Reference” button. Text with references, or hypertext, is in blue with a click affordance displaying the relevant context (i.e., highlighting elements (b1) replaying animations (D)). The process of the animation (inbetweening) is also displayed (F). The user can also filter the animation list (D) to only display the animations related to the selected elements.	64
5.3	Three screenshots of Jerry’s current user interface at different scrolling percentage.	65
5.4	Five screenshots of Jerry’s desired user interface at different scrolling percentages.	65
5.5	Five screenshots of the five actions that Jerry took to record his base scene.	66
5.6	CoCapture highlights the element, the profile image, that the cursor hovers over.	67
5.7	CoCapture allows users to link either an animation or a DOM element to part of their text to help make the description more useful.	68

5.8	For each created animation, CoCapture provides information that prompts users about the actions they can take, including creating visual tags and replaying or previewing the animation. It also supports a set of operations to make the creation process easier and more accurate, including a range slider for time adjustment, as well as options for cloning, changing, or deleting an element. . . .	69
5.9	There are three steps to create hypertext: (1) select portions of the text in the question description, (2) click “Link to A Reference,” and either (3.1) select an element in the scene or (3.2) check an animation. After this, the hypertext will be highlighted. . . . .	69
5.10	Time spent (s) on visual reference creation and text writing for the two conditions. The participants spent more time creating animations and less time writing the textual description in the treatment condition. For both cases, the differences were significant ( $p < 0.0001$ ). . . . .	74
6.1	A screenshot of a participant making a request to the hypothetical assistant while working on the hardware part of the embedded system project during the study.	86
6.2	Control web UI. It includes four parts: 1) a natural language task description, 2) a code editor that includes the task-relevant code, 3) two fixed-position live stream camera views, and 4) a 6-slider control panel that corresponds to the 6 servos of the robot arm. . . . .	87
6.3	A photo of the robot arm performing a wiring task . . . . .	88
7.1	Diagram of the on-demand expert programming support paradigm. . . . .	105
7.2	Screenshot from 2018 of some curated content from users’ posts in Snapchat, e.g., “So Satisfying: The oddly satisfying cut of this balloon...” . . . . .	110

# Abstract

In programming, *on-demand assistance* occurs when developers seek support for their tasks as needed. Traditionally, this collaboration happens within teams and organizations in which people are familiar with the context of requests and tasks. More recently, this type of collaboration has become ubiquitous outside of teams and organizations, due to the success of paid online crowdsourcing marketplaces (e.g., Upwork) and free online question-answering websites (e.g., Stack Overflow). Thousands of requests are posted on these platforms on a daily basis, and many of them are not addressed in a timely manner for a variety of reasons, including requests that often lack sufficient context and access to relevant artifacts. In consequence, on-demand collaboration often results in suboptimal productivity and unsatisfactory user experiences.

This dissertation includes three main parts: **First**, I explored the challenges developers face when requesting help from or providing assistance to others on demand. I have found seven common types of requests (e.g., seeking code examples) that developers use in various projects when an on-demand agent is available. Compared to studying existing supporting systems, I suggest eight key system features to enable more effective on-demand remote assistance for developers. **Second**, driven by these findings, I designed and developed two systems: 1) CodeOn, a system that enables more effective task hand-offs (e.g., rich context capturing) between end-user developers and remote helpers than existing synchronous support systems by allowing asynchronous responses to on-demand requests; and 2) CoCapture, a system that enables interface designers to easily create and then accurately describe UI behavior mockups, including changes they want to propose or questions they want to ask about an aspect of the existing UI. **Third**, beyond software development assistance, I also studied intelligent assistance for embedded system development (e.g., Arduino) and revealed six challenges (e.g., communication setup remains tedious) that

developers have during on-demand collaboration. Through an imaginary study, I propose four design implications to help develop future support systems with embedded system development.

This thesis envisions a future in which developers in all kinds of domains can effortlessly make context-rich, on-demand requests at any stage of their development processes, and qualified agents (machine or human) can quickly be notified and orchestrate their efforts to promptly respond to the requests.

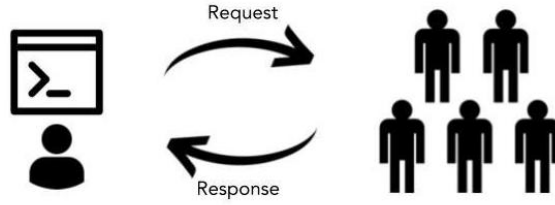
# Chapter 1

## Introduction

Programming is an expert task that requires complex reasoning skills. Support for recalling keywords, function names, argument types and methodologies, and other details can come from a variety of sources. Current support usually comes either from tools built directly into Integrated Development Environments (IDEs) or from other developers, including colleagues or online peers.

IDE tools provide contextualized, easily accessible, and on-demand support for developers, but they are generally limited in the types of feedback they can provide (e.g., error messages, syntax highlighting for error locations, and an auto-complete function) because the system cannot truly understand user confusion or the context of the problem. We find that these limitations preclude many questions that developers would ask while programming.

To overcome the limitations of automatic approaches, support from other human developers is often enlisted. This can take several forms, each with its own trade-off. In a work environment, an expert colleague can provide useful support regarding requests such as the specific languages and frameworks in use, but they are unlikely to be able to support frequent questions or to be available on demand. Web forums can provide a wealth of information about general questions, but they typically do not provide highly personalized support or quick responses to developer queries. Additionally, many of the most successful developer forums restrict the types of questions that can be asked and when they can be asked. For example, Stack Overflow [154] explicitly discourages the types of project-specific code questions that we observed in our motivating studies. It also requires some level of reciprocity from users in the form of useful answers to other developers' questions. Providing sufficient context for others to help solve a problem can be a time-consuming task that often takes multiple turns of interaction to complete.



**Figure 1.1:** Diagram of on-demand expert programming support paradigm.

Hired freelance developers can provide tailored support for specific questions, near real-time answers, and even the ability to hand off sub-tasks for independent completion. However, in addition to the monetary cost of hiring a freelancer, the traditional hiring process adds significant time and preparation costs (interviews, onboarding, etc.). Lastly, none of these expert solutions provide the in-context support that IDE tools do, adding the need for an additional context switching to and from a developer’s workflow.

The rise of online crowd platforms, such as Upwork [194] and Amazon Mechanical Turk (AMT) [9], allows the on-demand hiring of workers from a large online crowd. However, the complexity of the hiring and onboarding process to find reliable workers who have sufficient knowledge of the project remains unchanged. Reducing hiring overhead is critical to making expert support in development processes efficient. Ideally, asking for help would be as simple as informally stating the question at hand and receiving in-context on-demand support within an IDE. In this dissertation, I explore this help-seeking model (Fig. 1.1). Specifically, I explore the following questions:

- What are the needs and challenges developers face during on-demand collaboration with helpers who have no prior contextual information about the tasks?
- How can we design systems to provide effective on-demand support for various types of programming requests?

## 1.1 On-demand Collaboration With Expert Crowds

Throughout this dissertation, the term *on-demand* or *on-demand collaboration* will be used frequently. On-demand collaboration has always been a common behavior within teams and organizations. It occurs when an individual (“requester”) creates a task and sends it to one or

more people (“helpers”) who can perform the given task on time. Examples might include a manager asking their secretary to book a flight, a student asking their teacher for help on an assignment, or a developer asking their UI testing team to test the company’s website.

The process for this collaboration often involves two parts. The first part is coordinating tasks and helpers, including finding helpers who are able and available to perform the task. The second part is communication between requesters and helpers, such as describing the tasks and responses accurately and clearly. The goal of the collaboration is to fulfill requesters’ needs when completing the task. The AnswerGarden series of work has shown that providing on-demand support from experts can help organizations to maintain their knowledge database and increase overall work productivity [1, 3].

On-demand collaboration falls within the category of on-demand help-seeking. Since Web2.0, people have started to seek on-demand support not only from internal helpers within teams and organizations, but also from external helpers. For example, the emerging on-demand services on mobile applications such as Uber and Task Rabbit match real-world tasks with capable and available helpers, offering more responsive and streamlined courses of action and increasing market efficiency for microtasks [190]. Meanwhile, many community question-answering (CQA) websites have provided crowd expert support in the virtual world, allowing posted questions to be seen by a large community.

In the context of software development, this collaboration occurs when a requester creates a programming-related task and sends it (or has it sent) to one or more helpers. This thesis focuses on the case in which the requesters are programmers in need of support that can be more efficiently provided by remote expert developers than existing methods. These remote experts (helpers), on the other hand, are not within the same team and do not have prior knowledge of the task, but they have the expertise and availability to offer support. Unlike Stack Overflow where the response time is often too long (more than 11 minutes [132]), this thesis focuses on near real-time assistance for on-demand requests, which requires more effort for coordination and better communication support. Unlike AnswerGarden [1] where the help-seeking is within an organization (meaning there is less context switching and less need to communicate context), this thesis focuses on a more general-purpose support model in which helpers are outside of the organization, have not yet built up trust with requesters, and do not have prior knowledge of the task or requesters. This makes capturing and presenting the task context challenging in terms of effective communication between requesters and helpers.

Beyond software development, I have also explored this remote assistance model in many other contexts, including education [43], video analysis [45], and script writing [41]. This thesis focuses on developers' communication and coordination, which is important in all of these use cases, regardless of team size or incentive. The discussed issues in this thesis fit into the more general model that it advances.

In summary, the on-demand collaboration that this dissertation focuses on shares the following constraints and characteristics:

- **Collaboration model:** One requester to many helpers (or one-to-many)
- **Collaboration goal:** Increase developers' productivity (e.g., complete more coding tasks)
- **Relationship between requester and helper:** These groups do not know each other
- **Helpers' knowledge of context:** No relevant context before receiving the task
- **Distance between requester and helper:** Remote
- **Time:** Both synchronous and asynchronous
- **Types of tasks:** Tasks vary in both size and level of expertise needed in the context of software development

In the next few subsections, I will briefly describe four parts of my thesis work:

- two motivational studies that a) reveal challenges and needs that developers encounter when requesting or providing programming support, and b) motivated the design of this paradigm to provide on-demand, context-rich, expert support for software developers [40, 39],
- CodeOn, a system that allows developers to request assistance online as easily as they can through in-person one-on-one communication and tracks helpers' responses directly in the developer's IDE [42],
- CoCapture, a system that enables interface developers to easily create and then accurately describe UI behavior mockups, including changes they want to propose or questions they want to ask about an aspect of the existing UI, and
- two exploratory studies that 1) reveal the current practices, issues, and needs that embedded system developers have during remote collaboration, and 2) suggest design implications for on-demand programming support for remote embedded system development in the future.



## 1.2 Two Need-finding Studies

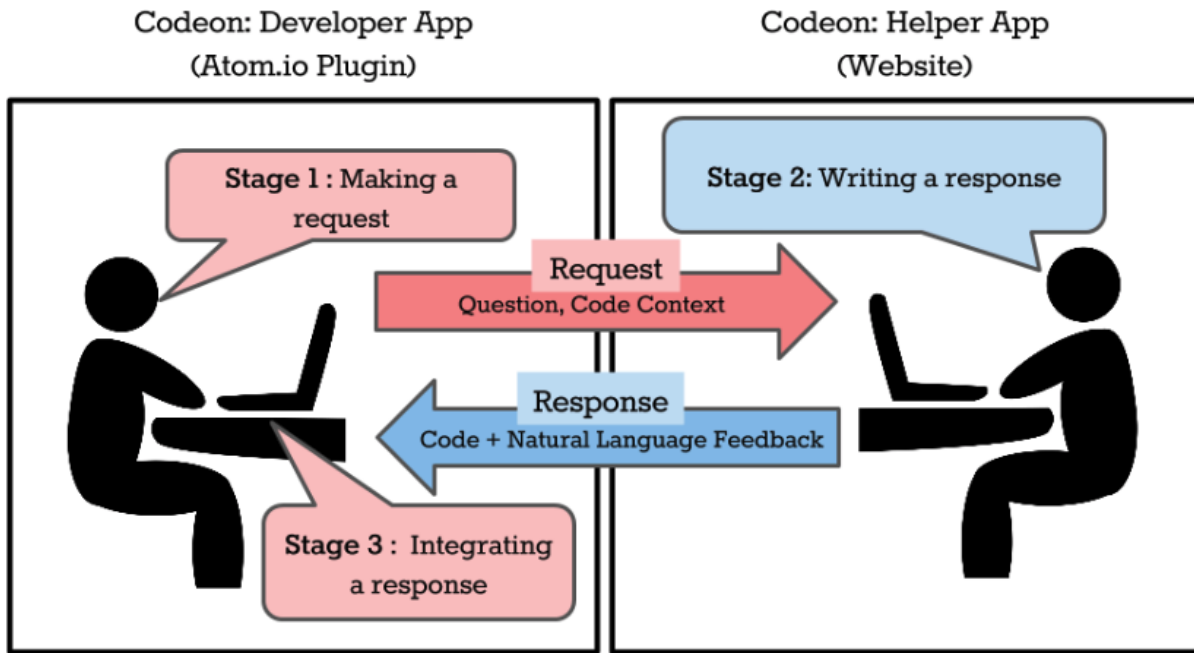
To motivate the on-demand expert programming support paradigm, we first conducted two studies that seek to inform the design of future systems that leverage remote experts to support developers on demand.

The first explores what types of questions developers would ask a hypothetical assistant capable of answering any question they posed. We found seven types of requests that developers would make to receive help from the assistant (e.g., high-level guidance, code refactoring, effort-saving requests where participants hand off tasks to delegate work.) The second study explores the interactions between developers and remote experts in supporting roles. Seven information needs have emerged from our observations and data (e.g., requesters did not like switching apps and would instead prefer a single, integrated system). Lastly, we suggested eight key system features needed for on-demand programming assistants to be effective, which has implications for future human-powered development tools. The findings from these two motivational studies provided guidance and research direction throughout my dissertation work.

## 1.3 Designing On-Demand Programming Assistance

Driven by the insights obtained from the two motivational need-finding studies, I then led the development of a collaborative programming support tool called CodeOn to instantiate the on-demand expert support paradigm (Fig. 1.2). CodeOn enables more effective task hand-offs (e.g., rich context capturing) between developers and remote helpers than existing synchronous support systems by allowing asynchronous responses to on-demand requests. One key objective of CodeOn was to minimize developers' efforts to complete their coding tasks by providing them nearly real-time personalized support and enabling them to control their own workflow by deciding when they want to interact with helpers. Another important goal was to create a UI to simulate an ideal programming support scenario in practice—one in which a developer can get assistance from a co-located helper simply by speaking and pointing to content on the screen.

CodeOn was tested via a series of validation tests exploring the trade-offs between the understandability, speed, and accuracy of each technique for different system components [39, 40]. For example, CodeOn supports voice requests, and the audio recording of the request is synchronized with developers' interactions within the editor (e.g., highlighting, scrolling, file switching) and can be replayed in the helper's interface. The multimedia request jointly embodies



**Figure 1.2:** Asynchronous interactions between developers and helpers can occur in three stages: making a request, writing a response, and integrating the response.

the dynamics of voice signals and visual content references, providing a rich, natural context for communication. My approach proved successful in the domain of software development, but it can be generalized to many other domains, including graphic design or UI prototyping, in which there exist novel challenges in finding dependencies, conflicts, and content relationships.

## 1.4 CoCapture: Effectively Communicating UI Behaviors on Existing Websites by Demonstrating and Remixing

While CodeOn can effectively support requests related to code syntax, many other requests involve visual information, such as those related to UI behavior issues. Unlike text (e.g., code syntax), visual information can be labor-intensive to describe. However, this information is often necessary in practice, as developers use it as context to ground their communication. In the context of UI behaviors, developers often try to communicate their needs by creating and annotating new visual references to explain the desired edits. However, creating UI behavior references can be time consuming, and they can be difficult to effectively annotate, making the requesting process more challenging. For example, a developer may screenshot a website, circle

a table in the image, and ask how to make it responsive. Similarly, a game developer may use sketch tools to create a basic illustration of a Super Mario game in an effort to demonstrate how the character should enlarge once it collides with a special object.

To address the challenge of communicating about UI behaviors, we first conducted two need-finding studies. From the results, we discovered that although developers often include visual context in their questions, many of these questions lack critical information needed for helpers to understand. We introduce CoCapture, a system that allows developers to easily create and describe visual information. We achieve this by enabling developers to mock up their desired UI behaviors on an existing website through direct manipulation, and to link the resulting mockup to their question. Specifically, CoCapture supports three intertwined activities: recording existing UI behaviors by capturing DOM changes, mocking up new behaviors as animations over the DOM-change recording, and linking these behaviors to a natural language question. Two lab studies with experienced UI developers showed that, when compared to existing communication tools, CoCapture helps developers create clearer questions that include 77% more information. Our approach can help develop software or prototypes more quickly, not only by helping developers to more accurately express their UI behavior vision, but also by reducing the information processing effort needed when reviewing questions.

## **1.5 Supporting Remote Collaboration for Embedded System Development**

Rather than being limited to software-only use, on-demand collaboration also occurs in other contexts, such as embedded system development (e.g., Arduino). Although embedded system development also involves coding, remote helpers face the additional limitations of inspecting and manipulating the remote circuit boards and other physical artifacts. To explore how to provide on-demand support in programming beyond just software support, I conducted two studies—an interview and a hypothetical assistant study—to better understand developers' current practices, issues, and needs during the on-demand help-seeking process. Besides being unable to allow remote collaborators to directly manipulate the physical artifacts, developers also reported six challenges (e.g., communication setup remains tedious) during their on-demand collaboration. The hypothetical assistant study revealed seven types of requests (e.g., physical aids: participants asked for physical activity support) that developers use when an on-demand

agent is available. Based on the interviews and study findings, I drew four design implications to help design on-demand programming support for remote embedded system development in the future.

## 1.6 Thesis Statement and Contributions

Through these projects, I aim to prove the following thesis statement:

**By understanding the needs and challenges developers face during on-demand collaboration with helpers who have no prior information about the tasks, it is possible to create intelligent systems that help coordinate and scale the collective effort of both parties.**

The core novelty that underpins my thesis work is an element-based recording and manipulating technique. It not only preserves users' ability to capture visual or audio information that is similar to the traditional video recording and replaying techniques, but it also preserves the underlying structure of interface elements such as text or DOM elements, as well as the interplay among these elements. Compared to pixel-based recording (video), this technique records users' interactions with the applications such the code editor and the web browser. Doing this effectively makes the recording into a sequence of different application states. Additionally, my work adds tools that allow users to easily manipulate these states to refine their questions or directly provide responses by building on top of the element-based recordings.

Overall, this technique brings three benefits: 1) It helps requesters more quickly and naturally record the process of task creation by piggybacking on the existing artifacts (e.g., code, user interface). It also enables them to easily modify the task by directly manipulating the recording, which in turn makes tasks easy to understand and reduces communication delay by making the task descriptions more accurate and comprehensive. 2) It also helps helpers to easily write their responses by directly manipulating or annotating the captured elements, providing more spatial and temporal guidance. 3) It makes helpers' responses more easily for requesters to understand and integrate into their workspace (e.g., codebase).

My dissertation contributes a set of new understandings and techniques for the on-demand collaboration paradigm in software development domains. Specifically, the work has made the following contributions:

- Study results that motivate the new paradigm of on-demand remote expert support for software development.
- A system (CodeOn) that instantiates the on-demand remote expert support paradigm and demonstrates its effectiveness in improving developers' task performance.
- A system (CoCapture) that extends this paradigm to make on-demand support related to UI behaviors more effective.
- Study findings that reveal the challenges that embedded system developers have during remote on-demand collaboration, and design recommendations to make this collaboration more effective.

## 1.7 Thesis Outline

Chapter 2 reviews the related work. Chapter 3 covers the two studies that motivated the paradigm of on-demand help-seeking from remote workers. Chapter 4 describes the design and implementation of CodeOn, evaluates its usability, and describes its approach in further detail. Chapter 5 talks about CoCapture, which focuses on making UI behavior communication more effective. Chapter 6 talks about the two exploratory studies to understand remote on-demand collaboration in embedded system development. Lastly, Chapter 7 concludes the thesis, summarizes the main takeaways, and discusses future work.

# Chapter 2

## Related Work

This thesis relates to three main fields: 1) collaboration in software development, 2) human support in end-user programming, and 3) remote collaboration for embedded system development. In this section, I review the papers that are representative of these three fields.

### 2.1 Collaboration in Software Development

#### 2.1.1 Community Question Answering

Many Community Question-Answering (CQA) websites, such as Stack Overflow [154], provide programming support that allows software developers to post questions to a large community. These sites also accumulated these questions and answers that they received to form a large database of questions and answers for later reference. Prior work [2, 3] studied building an “organizational memory” through a growing database of questions and answers. These CQA sites have a number of limitations such as a long waiting time to receive an answer after posting the requests and the fact that answers are not personalized. In addition, it takes significant time and effort to compose a question with enough context and explanation for other programmers to be able to provide an answer [11]. CodeOn enables speech and content selecting modalities, and also provides on-demand expert support that allows developers to describe the requests as if the helpers were physically nearby. They can select a code snippet, verbally ask “what does this mean?”, hand off execution of planned coordination to the system, and receive a meaningful response within minutes.

### 2.1.2 IDE-Integrated Help Finding Tools

CodeOn is a kind of Recommendation System in Software Engineering (RSSE) [170], which, unlike most CQA websites, often provides relevant information within an IDE. Prior work on RSSEs has used knowledge of how developers seek information to develop systems to provide semi-automated support [21, 22]. Blueprint [20] allows developers to rapidly search for a query in an embedded search engine in their local IDE. Seahawk [163] heuristically filtered search results to automatically increase the reliability of search results within an IDE. Hartmann et al. [79] also explored ways to aid developers in recovering from errors by collecting and mining examples of code changes that fix errors. These in-IDE approaches allow developers to save time by minimizing the change in task context associated with requesting information. Recently, Chen et al. [40] found that even with the state-of-the-art communication tools, such as Skype and JSBin, developers and helpers still face communication challenges when it comes to integrating answers into a codebase.

Tools that support developers using the crowd provide a way to potentially receive more personalized feedback than automated systems can do. CrowdCode [117] allows developers to make requests to the crowd with self-written specifications of the desired function's purpose and signature. But this approach is limited in how much it can reduce developers' time expenditure since making a request requires a detailed problem specification. Real-time crowdsourcing techniques have enabled on-demand interactive systems, which have been shown to be able to improve the efficiency of accomplishing complex tasks [116, 113, 114].

### 2.1.3 Programming Team Communication

Despite their reputation for preferring solitude, software developers who work in teams can spend up to half of their time communicating [81]. Team organization and communication mediums play an important role in the effectiveness of communication between developers. The model of task delegation system we propose in this paper is related to Harlon Mills' idea of a "surgical team" development model where the developer with the most experience with a particular task delegates more mundane tasks to other developers [141, 24]. On-demand code assistants would enable developers to dynamically create such supporting roles without the need to be part of a formal organization.

## 2.1.4 Pair Programming

CodeOn is related to pair programming [50], a method that allows developers to work together in real-time more effectively. In particular, it is most related to distributed pair programming, which is a derived version of pair programming, allows remote participants to contribute to the same codebase simultaneously [13, 173]. Although the distributed pair programming approach removes many issues in real-time remote collaboration [148], it can still be difficult to coordinate and maintain context in distributed pairs. Our system instead aims to automate coordination by temporarily incorporating helpers into a task long enough for them to assist and then move on.

## 2.1.5 Information Needs for Developers

Researchers have summarized the types of questions that developers ask in different contexts. Sillito et al. categorized 44 types of questions developers ask when evolving a large codebase [176]. Ko et al. explored six types of learning barriers in programming systems for beginners and proposed possible solutions from programming system sides [103], and also documented communication among co-located development teams [102]. Guzzi et al. analyzed IDE support for collaboration and evaluated an IDE extension to improve team communication [78].

Whereas these studies of information needs focused on existing team structures, our paper introduces a new path for information seeking via on-demand expert support, and the studies present qualitatively different data and implications. Unlike existing team structures, our paper proposes a team structure where a project stakeholder requests remote help from experts who are not stakeholders. This difference has significant implications for team trust, communication preferences, and context sharing.

## 2.1.6 Collaborative Development

Systems like Codeopticon [72] and Codechella [73] provide ways that helpers (i.e., tutors/peers) can efficiently monitor the behavior of multiple learners and provide proactive on-demand support. Version control systems such as git are often used in programming collaboration because they help developers in distributed teams synchronize source code. However, version control systems also require that developers manually push and pull changes and resolve merge conflicts. Collabode [67] introduced an algorithm that addressed the issue of breaking the collaborative build without introducing the latency and manual overhead of version control.



CodeOn fetches developers' latest code before helpers can send their code responses to allow more experienced helpers to resolve merge conflicts.

Communication tools like Slack or Skype make collaboration more effective by supporting conversational interaction, but it is often challenging to capture the code context within these tools. Commercial IDE tools such as Koding [177], and Cloud9 [89] enable users to code collaboratively online in real-time. Although these systems reduce many of the barriers developers face when working at a distance [148] and time spent on environment configuration, they do not support the case when developers are actively seeking help [182]. CodeOn allows developers to create requests at any time by speaking their questions while the system automatically captures the problem's context.

## **2.2 Human Computation in Software Development**

Most work has investigated how to improve crowd workflow by converting parallel tasks to series tasks [173], and letting the crowd workers guide the workflow [128]. However, little work has focused on exploiting expert crowds to help with complex tasks such as programming in real time. The design findings presented in this paper will help guide systems that can coordinate crowd experts in real time.

### **2.2.1 Real-Time Crowdsourcing**

The model and features I introduce in my thesis rely on the ability to quickly recruit experts in order to improve question response time. Pera and Ng have shown that CQA sites can improve their algorithms for question-matching, which would help developers find relevant archived answers [161]. Another way to improve the response time of CQA sites is to route questions to appropriate experts, as Riahi et al. propose [169]. However, both of these techniques still require that developers spend time to carefully formulate their questions while including the relevant contextual details.

Instead, we propose building on previous techniques for real-time crowd recruitment. VizWiz [17] introduced a model for eliciting real-time responses from the crowd by keeping workers engaged in example tasks until their assistance is needed in real time. This model can elicit responses from the crowd in a matter of seconds. Adrenaline [16] takes a similar approach by queuing idle workers so they may complete other tasks while they wait. LegionTools [115] is the

first task-independent open-source tool for recruiting and managing real-time crowds. Legion extends the real-time crowdsourcing model to include continuous tasks — tasks that span as long as the worker chooses to stay engaged — over small, disjoint microtasks. This helps retain context and greatly improves answer latency.

Aardvark improves its response time by routing questions to experts who are currently online [85]. Chorus [114] enables on-demand conversational interaction by recruiting multiple workers for conversational interactions with users. Apparition [113] enables prototyping interactive systems in real-time by introducing a self-coordination mechanism to reduce task conflict among workers.

By combining these techniques with an IDE-integrated communication mechanism, future tools can enable quick and meaningful responses to software development questions.

## 2.2.2 Crowd-Enabled IDE Tools

A few tools have enabled crowds to aid in development tasks. Latoza et al. [117] developed CrowdCode, which decomposes programming into self-contained function-based microtasks. In CrowdCode, clients make requests to the crowd with self-written specifications of the desired function’s purpose and signature. Crowd workers are then automatically assigned to these tasks. However, such a system is limited in how much it can reduce the end user’s time expenditure, since the request authoring process requires a detailed problem specification.

Having the crowd to efficiently assist programming has been challenging in terms of qualified worker recruitment and efficient work evaluation. Collabode [67] allows users to define function-based microtasks, including testing and debugging a function, which allows workers to evaluate the previous work. It also provides workers the choice of skipping the microtasks which do not match with workers’ skill sets. We propose recruiting helpers and allowing them to choose the tasks, allowing users to decide whether the answer is valuable.

## 2.2.3 Expert Guidance

Beyond CQA websites, several commercial systems enable more personalized mentorship for software developers. Code Mentor [90] and hack.hands() [92] allow software developers to create requests that connect them with experts (as judged by self-report and community reviews) quickly. Help requesters and experts can communicate through a shared code editor, chat window,

and video chat. Based on the findings of our studies, we propose a “per-question” assistance model rather than the mentorship model introduced by these systems. In a per-question model, requests for help can be asked and answered asynchronously rather than requiring the developer to start a new help session for every question. Further, this model would allow the question to be routed to multiple potential experts; if one does not know the correct answer, another can respond. This is unlike the mentorship model, where if a code mentor does not know the correct answer to a question, the developer must initiate a new session. Finally, the “per-question” model would enable better expert selection because unlike the mentor model, the question itself would be known before connecting the help requester and expert.

Beyond AnswerGarden (discussed above), several systems have proposed systematically routing help requests to an appropriate expert. Expertise Recommender [135] and SHOCK [130] both include mechanisms to route questions to users who are capable of answering them. Expertise Recommender relies on collaborative filtering to build expert profiles, whereas SHOCK automatically builds expertise profiles based on tasks the user has performed on the Web. The designs of these systems may help guide the question-routing architecture for our proposed question-routing system.

In the domain of software development, Mockus and Herbsleb proposed Expertise Browser [142] as an approach to find experts in the context of a shared project. It relies on repository commit data from prior projects to quantify expertise in the context of geographically distributed development teams. However, we propose creating systems where software developers can seek guidance from outside of a project team, which requires a different method of determining expertise.

## 2.2.4 Expert crowds

In this thesis, we use crowdsourcing methods to make remote assistance available on demand and scalable. By using expert crowd platforms like Upwork [194], which have thousands of developers with a wide range of expertise, we can hire as many experts as needed, with a wide range of expertise, to fill a developer’s set of requests. This allows CodeOn to parallelize as much as the end-user developer may want to.

Although promising, there are three main challenges when working with experts on crowdsourcing platforms. First, expert matching can be challenging. Finding a qualified and available expert that can answer a question can greatly reduce the number of iteration especially when the

response from the CQA system does not satisfy the asker [129, 203, 204, 169].

Second, once matched, experts need to quickly and clearly understand the requests. However, composing a question with enough context and explanation often takes significant time with existing approaches [11]. CodeOn enables speech and content selecting modalities, and also provides on-demand expert support that allows developers to describe the requests as if the helpers were physically nearby. They can select a code snippet, verbally ask what it means, hand off execution of planned coordination to the system, and receive a meaningful response within minutes.

Third, coordinating and managing all the requests, their status, and the experts can be challenging. Prior work has explored how to use prior tasking and guidance to automate the coordination and task management process. Foundry [168] provided an interface for composing expert workflows for large tasks. Foundry was used to create *Flash Teams*—dynamic, expert crowd teams—to complete tasks faster and more efficiently than self-organized, or crowd-managed groups. In our work, we focus on similarly-focused tasks with well-scoped hand-offs, but do not assume that developers know the high-level composition of tasks in advance, instead allowing developers to define tasks on-the-fly as they discover and generate them.

## 2.3 Communicating UI Behaviors

As Myers et al. [144] explain, interactive behaviors define the “feel” of a UI (as opposed to its “look”). Tools like VisBug [68] and Poirot [188] help designers quickly change the look of their UIs. We focus on communicating about the feel of the UI. This process often consists of two tasks: making visual references and referring to the visual references. In this section, we review related work and techniques in these fields.

### 2.3.1 Creating UI Prototypes and Mockups

Systems like SILK [112] and DENIM [127] lower the overhead cost of prototyping by recognizing designers’ sketches as interface elements and implementing the idea of wireframing, respectively. However, they do not support creating prototypes in later stages of UI development. More recently, tools like Rewire [185] and Poirot [188] made prototyping new designs easier by enabling the users to directly edit elements on existing examples. However, they do not support interactive UI behavior editing, which is more difficult than designing static layouts,

as the behaviors are complex to demonstrate and designers have access to limited tools [157]. Commercial tools (e.g., Figma [57], Adobe XD [4]) can ease the creation of interactive behaviors but assume their users would reconstruct existing interfaces from scratch, making it hard to scale. Crowd-powered systems like Apparition [113] and SketchExpress [121] allow designers or even non-experts to more rapidly create or reconstruct a prototype than they could with existing tools, but these systems also fall short of recreating particularly complex interfaces.

Unlike these systems, CodeOn helps create interactive mockups in the later stages of UI development, proposing changes to a UI that already works or describing desired behaviors in a UI that contains an error. These UIs can be arbitrarily complex, requiring hours of work to create mockups that replicate existing functionality. By capturing the DOM structure, CodeOn allows designers to easily add behavior mockups on existing interfaces, giving them the ability to immediately envision the new behaviors and complete the process of creating a mockup as a reference.

### **2.3.2 Visual References as Shared Context in Communication**

Many prior studies have reported that people often include screenshots, drawings, or sketches as visual context in communication. These studies have explored questions of how designers communicate desired interactive behaviors to engineers [144, 157], how InfoVis novices describe data visualization [139, 69], how programming novices explain PC game behaviors to the computer [155], and how end-user developers communicate about application extensions with other developers [52]. However, they also consistently found participants' responses to be vague, ambiguous, or imprecise, suggesting future systems should provide a tight feedback loop in which users see immediate results to refine ambiguous description.

Creating visual references can help ground communication when discussing visual design and providing feedback. In a face-to-face or video conference setting, we can use pointing gestures in shared visual spaces to make references to visual information that is difficult to express with words. Much work has studied methods for referring to visual content in communication, including text annotation [146], remote gestures [60, 98, 100], and awareness widgets [55, 75] in different computer-supported cooperative work contexts such as authoring [55, 136, 198, 201, 202, 172] and groupware [60, 76, 77]. They have shown that referring methods can facilitate mutual understanding by reducing verbal effort and its associated complexity [75]. The ability to leverage non-verbal communication is an important factor in decreasing the effort of writing

clear messages [49].

In programming communication, systems like chat.codes [150] and Callisto [196] use deictic code pointing techniques to facilitate creating code references and connections with text descriptions to help developers discuss code. Codeon [42] built an in-IDE support environment to help requesters and helpers exchange code context easily. MarmalAid [47] allows users to start a real-time conversation on a geometric location in a 3D workspace. Building on these approaches, we aim to address the problem of referring to dynamic and interactive visual references for more effective communication.

### 2.3.3 Record, Replay, and Manipulate Existing Interfaces

A core technical part of CodeOn’s system is the record and replay (R&R) technique, which is used to record an existing behavior once and then replay it repeatedly and automatically without user interaction. Prior work has used this technique for various purposes. Systems such as Scry [26], Telescope [82], Unravel [83], Doppio [46], and WebCrystal [31] use this approach to help people understand existing UI behaviors. Systems like Chronicle [71], Timelapse [27], and MobiPlay [165] record meta-data (e.g., operations, code editing) and allow users to easily capture the rich data of application behaviors. Our techniques enable designers to not only record arbitrary web interface behaviors, but also to easily add new designs on top.

To ease the creation of new behaviors, FrameWire [126] decreased the effort of communicating new interactive designs by automatically extracting interaction flow from paper prototype video recordings. Many other recording augmentation systems have been developed to help effectively prototype new digital content, such as Montage [123] and AR experiences like Proton [124], or to provide video feedback, like VidCrit [158]. Park et al. developed a technique that enables users to edit text content in a text-based recording while preserving the recording’s overall consistency [156]. CodeOn also allows its users to easily add new behaviors over recordings, but instead of simply supporting overlaid visual annotations or user comments anchored to specific parts of the content, it allows users to edit existing elements through direct manipulation.

The field of **PbD!** (**PbD!**) has used similar techniques to augment users’ ability to perform tasks with which they often lack expertise. Rousillon [34] and Sugilite [125] allow their users to record and edit the operations via domain-specific languages. CodeOn also supports a set of mockup creation operations that allow designers or even novice users to edit the recording and simulate the desired interactive behaviors.

## **2.4 Remote Collaboration for Embedded System Development**

In addition to software development, other engineering or system development work includes tasks that require collaborators to operate physical artifacts in a physical environment, such as embedded system development. When collaborating on these tasks, having a shared workspace is valuable because it helps ground conversation in shared understanding [58, 66]. However, collaborating with other people has become as likely to take place over distance or time as it is face-to-face. In particular, with more organizations encouraging their employees to work from home, in addition to a growing market for freelancers, remote collaboration has become a necessity for many workers, developers, and clients, a change that can be problematic.

Exploring and studying effective methods to help people remotely collaborate on physical tasks is an important topic in the field of CSCW. Prior work has shown that remote coordinators are adept at adapting their communication strategies for nearly any type of shared information [107, 66]. In this section, we will review various remote collaboration techniques for physical tasks.

### **2.4.1 Remote Collaboration During Physical Tasks via Audio/Video Techniques**

Audio and visual information are the two most important types of information to exchange during communication. Techniques that provide audio and visual information to long-distance collaborators have been available for decades. Many prior studies have looked at how these techniques affect the efficacy of communication, the manner in which collaborators communicate, and the coordination between people (e.g., who leads the task). For example, early work on remote collaboration using these techniques during mechanical repair and maintenance tasks has shown that audio-only communication is not as efficient as audio/video communication [58]. However, audio/video communication is still less effective than co-located communication. Researchers have also argued that depending on the type of tasks, collaborators may require different types of visual information to help them communicate [58]. Better systems should be developed to meet these types of collaborative working needs.



## 2.4.2 Support for Remote Collaborative Tasks via Augmented Reality Techniques

More recently, new techniques have emerged to provide more “present” types of collaboration that make the process more effective. Augmented reality technology has the potential to change the ways we communicate and collaborate by augmenting the real world with virtual information [205, 195]. Augmented Reality (AR) technology has applications in many domains, including planning, gaming, and education [205, 195], but one of the most often cited use cases for AR is *remote guidance*, where a remote *helper* assists a *worker* in performing a physical task. Remote guidance is valuable in a variety of situations, particularly for tasks where workers need to reference helpers’ expertise [80, 191]. AR has the potential to improve communication in remote guidance tasks by allowing helpers to augment a worker’s physical space, adding instructions to help them perform a given task.

Previous literature has proposed two primary ways for helpers to augment a worker’s physical space to give remote guidance: *annotations* (e.g., [97, 147, 131, 35, 153, 63, 60, 32, 54, 56, 64, 74, 184, 65, 186]) and *telepresence* (e.g., [6, 10, 15, 111, 7, 86, 87, 180, 118, 61, 62, 5, 199, 110, 179, 181, 189, 99, 59, 187]). Annotations are virtual notes or marks that remote helpers leave in a worker’s space as guidance, augmenting the environment with expressive instructions [74]. Telepresence represents all or part of the helper’s body virtually and displays that representation in the worker’s space [183]. This allows helpers to communicate through a rich vocabulary of gestures and deictic references [159, 138, 105].

## 2.4.3 Remote Collaboration for Embedded System Development

While these AR techniques have shown promise, the tasks that prior work has focused on were mechanical in nature, such as bicycle repair. Embedded system development tasks have four differences from mechanical tasks. First, mechanical tasks often have a ground truth, meaning that when a requester asks an expert for help, the expert often knows the desired output, such as making the bicycle function once more. However, this is often not the case in embedded system collaboration. Makers often create new projects, and helpers may not always know the desired output ahead of time and might require more time and effort to reach common ground. Second, the information required for reasoning about mechanical tasks is often immediately visible. For example, with a furniture assembly task, each component and the progress toward the desired



goal is visible to the human eye. However, with embedded systems, not all information is immediately visible, such as currents that go through certain components or computer code embedded into the chips. This adds another barrier for people trying to reach common ground. Third, mechanical tasks have a fixed number of dependencies needed for reaching common ground. However, prior work has shown that during the development process, developers often face challenges integrating the software and hardware [18]. This makes the context sharing more dynamic and complex. Lastly, because of the first difference, in which mechanical tasks always have a ground truth known by helpers, the tasks also have a finite number of solutions with which workers can get assistance. However, because makers often are working on new projects, helpers may not always be capable of providing assistance. This in turn may add to the overhead cost of the collaboration (e.g., finding another helper).

To address these challenges, existing tools such as Bifröst allow developers to more efficiently debug embedded systems [137]; likewise, Heimdall allows instructors to remotely inspect, measure, and rewire students' circuits [95]. However, both tools have drawbacks: use of Bifröst is limited to specific tasks, and developers must still rely on their own capacity to complete their tasks. Heimdall requires students to build their circuits on a specialized workbench, limiting the size of students' projects and prohibiting them from working in parallel when seeking help [95].

#### **2.4.4 Teleoperated Robots for Remote Collaboration**

As teleoperated robot technology becomes cheaper, more powerful, and more reliable, remotely operated telepresence robots will become more prevalent in homes and businesses, allowing visitors and business partners to be present without the need to travel. When it comes to remote collaboration, prior work has looked at two aspects of using teleoperated robots to facilitate the communication and coordination. In this section, we review two of the relevant aspects: privacy and the user interface.

##### **Privacy**

Privacy is of the utmost concern in many remote communications; phone calls are often taken in quiet places away from loud noises or important discussion. Previous research has studied users' perceptions of privacy in various scenarios with teleoperated robots. Butler et al. explored the privacy-utility tradeoff when a remote teleoperator received different types of visual information from a fixed-position camera. They found that a small loss in utility on the remote operator's

side resulted in large privacy gains as perceived by the end users [30]. Hubers et al. studied how different video manipulation techniques affect privacy information exposure when remote operators explore pre-recorded video scenes [88]. Klow et al. also explored multiple real-time video-filtering techniques when remote operators control a robot exploring physical scenes. While they found that these techniques can effectively protect certain privacies, the tasks were too simple and did not require much detail or changing visual information (e.g., assembling furniture, wiring a circuit). We argue that more studies and techniques should be explored using more complex tasks that require rich information, such as object color or multiple, continuous steps for completion.

### **Teleoperated Robot User Interface**

On the operator's side, a robot's maneuverability also plays an important role in task performance. Most of the participants in prior studies were non-experts in robot control. If the filtering techniques mentioned above were applied, helpers might have difficulty navigating and controlling the robot. Prior work has explored different methods of helping operators to both see the physical environment and perform the tasks.

Multiple studies have shown that adding autonomous motion and planning to a robot arm can effectively help humans handle complex tasks. Leeper et al. studied how differences in the level of robot arm autonomy (i.e. a case in which the user needs to specify the target grasp position vs. one in which the user needs to click a pre-defined set of auto-computed grasp positions) affect a task's success rate. They found that people performed better when using strategies with higher levels of autonomous assistance [122]. Kent et al. also compared two different user interfaces for remote robot arm control and suggested that the point-and-click user interface, which proposes grasp poses based on local 3D surface geometry, can help users complete more tasks and reduce errors [96]. To enable helpers to operate remotely, future systems can adopt the target-tracking technique proposed by Rakita et al. [166], which uses a camera robot that automatically moves to provide a view with an appropriate camera-target distance. These studies showed promising results but only focus on simple tasks such as pick-and-place tasks. Future studies should explore various types of tasks, as found in our studies (Chapter 5).

# Chapter 3

## On-Demand Expert Support Paradigm<sup>1</sup>

To put the prior work mentioned above in perspective, and to illustrate the potential advantages of the features we will discuss in the following chapters, consider the case of Anita, a software developer working on a project that she would like to release as open-source software.

### 3.1 Motivating Scenario

Anita begins development using an IDE with on-demand support enabled. As she codes, the IDE reminds her of code blocks that need comments to be understandable to others.

After implementing the basic structure of the client for the tool she is building, she decides to integrate an Application Programming Interface (API) call to a service that she has never used before. Instead of breaking from her coding task to start looking into online documentation and forum examples, she requests (from her IDE) expert help to write the function that contains this API call, and asks that the response explain the implementation decisions and functionality so that she can better learn the API for the next time she needs to use it. While the operation completes based on a simple spoken natural language request, she continues to develop her tool as she otherwise would have.

When feedback arrives that her function is written, she checks it and sees that the support developer has left extra details about the implementation because it used a software design pattern that she had not used before very frequently. After understanding the code, she approves the automatic integration into her current code, and then resumes building her tool.

Later, when Anita is unsure of why a returned value is not being parsed as expected, she

---

<sup>1</sup>Portions of this chapter were adapted from [40]

makes another request to an expert to see if they know anything more about this particular library function. Within seconds, a support developer who is experienced with the frameworks and libraries being used in this part of the tool is looking at Anita's code and explains via chat that a segment of code that they have highlighted would need to pass an additional parameter to the function in question in order to get the right return type. Anita adds the fix and thanks the developer, all without ever leaving her IDE window. With her tool completed faster and with fewer distractions than she expected, Anita posts it online.

### **3.1.1 Comments and Discussion**

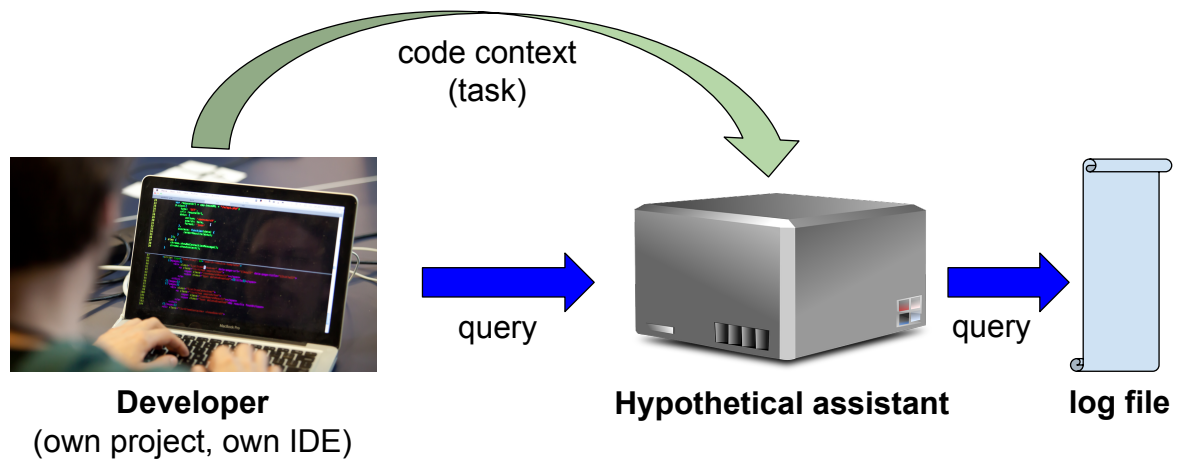
Since Anita does not have to interrupt her workflow to find information and delegate tasks, she is less distracted and more productive. She also does not have to lose time specifying tasks in complete detail. Instead, the system automatically extracts additional information based on code structure, previous documentation, and her interactions with the code. The ability to specify when the remote developer should provide additional details of a solution helps provide learning benefits when desired, while not wasting time or helper effort when the explanation is not needed. Additionally, because the system recruits experts on demand, Anita is only billed for the time and support that was actually needed. This is in contrast to the naive solution to providing on-demand support by keeping a single developer on retainer for the entire session.

## **3.2 Challenges**

Computers alone will not be able to provide the functionality described above any time in the near or medium-term future. The natural language understanding, program synthesis, and problem solving components of this system are each well beyond what is currently possible with AI alone, and the combined challenge is disproportionately harder to solve.

In order to overcome this, we need to leverage human intelligence and expertise. However, little is known about what people would actually choose to ask if the scope of support tools was not limited a priori by the system, or what challenges human experts face when asked to provide on-demand support. Increasing understanding of these two aspects of the design space will allow system builders to better reason about the challenges that need to be overcome to create fast, usable, and efficient assistance tools for software developers.

To understand how developers would use on-demand expert support, we observed how



**Figure 3.1:** Setup for our hypothetical assistant study. Developer participants were asked to bring their own task to complete, and ask questions from our hypothetical assistant as if it could answer any support question needed. Participants were still allowed to use traditional online resources and augment them with the assistant as they saw fit. Context related to their programming task was collected at the beginning of each study, and audio of their questions was recorded during the session.

developers interacted with a “hypothetical assistant”. Our use of a hypothetical assistant is similar to Shewbridge et al.’s use of a “faux 3D printer” to explore future uses of 3D printing at home [175]. It allows us to better understand the types of questions that software developers would want to ask an intelligent assistant in the “best case” scenario: if there are no limitations on its time, compensation, or capability.

### 3.3 Study 1 Design

We recruited five participants from the authors’ university, each with different levels of programming experience. We asked participants to work on their own programming projects on their computer and IDE in a laboratory for 45 minutes. We instructed participants to imagine that they had an intelligent human assistant nearby who is capable of answering any questions that they express verbally. Participants could use their hypothetical assistant to answer questions or perform a variety of types of work. The hypothetical assistant served as a conceptual prop that participants could make requests to, while we recorded audio, as Figure 3.1 illustrates. After the study, we conducted an interview regarding the hypothetical intelligent assistant and looked for commonalities.

## 3.4 Participants' Request Categories

We filtered out non-programming questions and found that participants' requests fell into seven categories. Table 3.2 shows the seven categories, the number of sessions in which each question type appeared, and their overall frequency. We explain each category in more detail in this section. We will reference these categories in the design section.

### 3.4.1 Memory Aids

In *memory aid* requests, participants knew semantically what they were looking for, but sought information regarding the exact syntax required. For example,

P1: "... I forgot how to use the syntax for `glmer()` function in R, I forgot like specifically how to write the random variable like to feed to the function, so can you find that out for me?"

P5: "What's the command to put in terminal in the command line when trying to make a file and executable?"

### 3.4.2 Explanatory Requests

In contrast to memory aids, *explanatory requests* seek explanations in addition to specific information.

P1: "I'm curious about what the `[a]` argument *actual* means or like does in that functions can you find that as well."

### 3.4.3 High-Level Strategic Guidance

Unlike the previous two request categories, *high-level strategic guidance* questions ask for the best way to approach a problem. Here, participants have a high-level idea of the task they wanted to perform, but were not sure how to translate their idea into code, or if it is possible.

P1: "I'm trying to like make a Bayesian network model based on the current existing data, and I just wonder like is there a good library or package that I can use from R..."

P3: "Wondering if there is a way to ensure to understand if the radio button in HTML can be color coded."

Description	# Sessions (out of 5)	# / Session
<b>Memory Aids:</b> Participants sought a specific function name	2	0.6
<b>Explanatory Requests:</b> Participants sought examples or explanations of their code	4	1.8
<b>High-Level Strategic Guidance:</b> Participants sought best ways to approach problems	5	4.6
<b>Code Requests:</b> Participants sought specific pieces of code	2	0.6
<b>Bug Fixing:</b> Participants sought specific solutions to program errors	2	1.0
<b>Code Refactoring:</b> Participants asked for code improvements	2	0.8
<b>Effort-Saving Requests:</b> Participants handed off tasks to save time and effort	4	4.0

**Figure 3.2:** Common query types observed during our hypothetical assistant study, with corresponding frequencies. Each of these query types suggests a support role that remote software development assistants can play in future systems. “Number of Sessions” indicates the number of different sessions that each type of question occurred in, while “Number of Queries per Session” indicates the number of queries that referred to solving one of these query types, on average.

### 3.4.4 Code Requests

Participants also asked for specific blocks or portions of code from the hypothetical assistant.

P3: *“How to make an AJAX get call from JavaScript”*

### 3.4.5 Debugging Requests

Two participants also requested help debugging their code. Although this type of request was relatively uncommon, we believe this might be due to our study’s short duration. Debugging examples are:

P1: *“I’m trying to make a new variable from R by selecting like some parts of the data columns, but I kept getting an error. . . is there a way to like which column name has been mistype, or doesn’t exist in the existing data frame?”*

P1: *“I’m trying to integrate MySQL database to R using R MySQL library I’m using the db connect function, but I keep getting an error if I use something like remote hostname other than localhost I’m not sure what’s the problem. [sic]”*

P1: *“I’m using [a] function from [a] package from R, I do . . . I’m getting unconstrained network result from, I want to constraint the number of network by 2, but sometime it gets too much time to compute then getting out unconstrained network which is very strange for cause’ I thought it would cost less time than the constrained one. Can you find out why? And suggested how should I fix my code?”*

### 3.4.6 Code Refactoring

Two participants also requested help *refactoring* their code to improve its structure. Many of these code refactoring requests also had an educational component; participants wanted to refactor their code to improve their coding style.

P1: *“I’m using ROC curve to evaluate logistic regression, not sure whether that’s a good idea or not, can you find other kinds of function that I can use to learn...?”*

P4: *“I’m setting up a variable method to manipulate the class I’m making in the init() function which I’m supposed to do, I’m also curious as to whether I have to make persistent variables elsewhere outside of init() method, whether I can make a new variable on self anywhere.”*

P4: *“What are the best like refactoring patterns going forward that could make a non object-oriented script.”*

### 3.4.7 Effort-Saving Requests

Four participants also made *effort-saving requests* to automate parts of writing and debugging code that they appeared to find tedious. Interestingly, a large number of effort-saving requests involved writing tests for code.

P5: *“Do some unit test for me to set key and get key.”*

P3: *“I can test it on IE, can you please test this on Safari? Because I don’t have a Mac machine”*

P4: *“Make some of the document that I’m testing, like I’m making a [a] document that’s kind of a pain to make, it will be great if someone can do that.”*

## 3.5 Findings and Interviews

Overall, each participant asked an average of 15 questions. Participants often phrased questions in a way that required knowledge of their code base. In other words, only 18% of participants’ questions were “self-contained”.

To better understand participants’ concerns and suggestions, we conducted a follow-up interview. In this interview, we gathered their opinions on the advantages, disadvantages, and desired features of a hypothetical code assistant. Every participant indicated that timely responses would be crucial to their adoption of such an assistant. Four participants also cited



personalized answers and free form questions as another potential advantage of such an assistant over current systems.

P1: *“A lot of time I like ask questions person to person. It’s much easier to articulate my questions. . . ”*

The most cited concern amongst participants was data privacy; two participants would prefer to have fine-grained control over what parts of their code they transmit when they ask a question. Three participants were also concerned about a potential lack of learning benefits compared to traditional sources of information.

P5: *“One bad thing that I think the system. . . goes against the very nature of computer scientists, because computer scientists need to solve those problems on their own”*

One participant, who is a non-native English speaker with a heavy accent, saw relying on voice input as a potential disadvantage. This participant found that they would prefer to type questions rather than speaking them.

Finally, we asked participants the media with which they would prefer an intelligent assistant responded. Although participants had a strict preference between text (3/5) and voice (2/5), most participants’ chosen medium depended on the type of response. One participant preferred a combination of the two: voice feedback with bullet point reminders. Participants also differed in their preferred style of response. Two participants preferred a “teach me” style of feedback, which they would translate into steps. However, two participant preferred a “show me” style of feedback when the question is short, with directed instructions.

We ran a second study to explore how developers try to on-board remote experts on demand. Unlike the hypothetical assistant study, our study focused on communication challenges, such as on-boarding challenges, contextual needs, and interaction difficulties.

## **3.6 Study 2 Design**

Whereas the first study referenced a hypothetical intelligent assistant (which we think of as a “best-case” assistant), our study used human assistants. We paired participants so that every session had one “requester” and one “helper”. We recruited 24 participants for 12 sessions, and gave requesters Web development tasks that involved displaying JSON data from a URL. Helpers were asked to respond to any queries from requesters (through voice and text) during the study. Figure 3.3 illustrates our study setup.

We determined whether each participant would be a helper or requester based on their Web programming experience. To measure this, we gave participants a pre-task survey of their JavaScript and HTML skills. We assigned participants who scored six or above (out of nine points) to helper roles and those who scored five or below to requester roles. This simulates the case where a future system is able to recruit the “right” helper for a job. Our experiments are intended to explore the interaction between helpers and requesters after the recruiting has been completed.

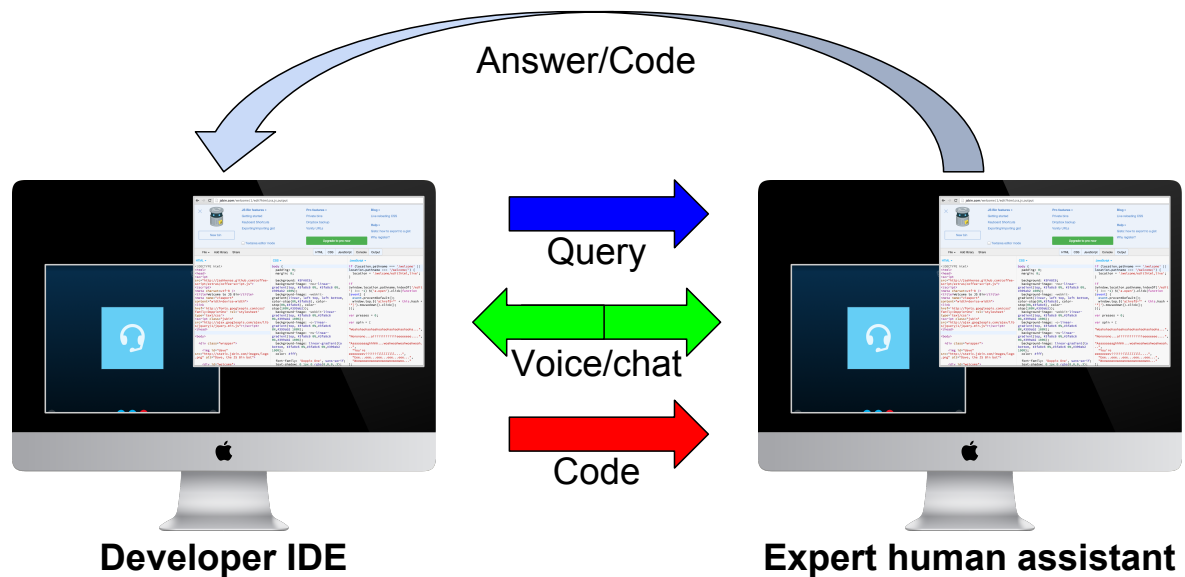
All 12 requester participants were students at the authors’ university, as were half (six) of the helpers. We recruited the other six helpers from Upwork [194], a popular freelancing platform. We specifically recruited Upwork participants whose profiles indicated that they were professional programmers. We used the same survey to ensure they were equally qualified to be helpers as our local participants.

During each session, requesters and helpers were physically separated, and requesters can ask for support from helpers via Skype, which connected them through the session. We asked requesters to share their screen in order to provide enough context to the helpers. We collected audio recordings during the study to capture the conversations that the pairs had, and the responses to the follow-up questions we asked them.

Unlike most pair programming paradigms, we specifically asked (and reminded them as needed) the helpers to be strictly *reactive*. This is in contrast to many pair programming paradigms where experts proactively monitor participants’ code. Instead, helpers would only look at requesters’ code when the requesters explicitly asked a question. This allowed us to better simulate the environment that intelligent code assistants would operate in: responding only to explicit requests for help and answering questions without having the complete context in which they were asked. This also allows for increased flexibility in the types of experts or crowds that can support such a system in the future.

We told requesters that they could also search for help using any standard Web resources (e.g., Google, Stack Overflow, or library documentation sites). By letting requesters use third-party resources, we gain a better understanding of the types of questions they prefer to answer through web resources versus where consulting a helper was preferred. After the study, we asked participants a few questions to get feedback on their experience with respect to the setting and helper, with specific examples of issues that arose during the study.

After analyzing the data, we generated a list of features related to participants’ concerns



**Figure 3.3:** Setup for our human expert assistant study. In each trial, one “requester” participant (developer) was paired with one “helper” (expert, based on a pre-study skill assessment). Participants could chat via either text or voice, and requesters were able to share their screen with helpers as needed using Skype. The helper was tasked with assisting the requester *reactively*, meaning that they only responded to queries, and did not proactively propose solutions or approaches. This simulates a “best case” (repeated, non-multiplexed helper) on-demand model where human experts are not expected to be continuously available between end-user queries.

during the study, and then re-contacted all of the 24 previous participants. We received 11 replies (5 helpers, 6 requesters). We asked for their feedback on how effective they think the suggestions would be, given their experience during the study.

### 3.7 Findings

To understand the experience that participants had during the study, we transcribed and analyzed all of the conversations and interviews from our audio recordings. Following the thematic analysis method, we first read through each turn of the requester-helper conversations, and the interview responses. This resulted in the identification of 6 distinct user information needs (see Table 3.4). We also counted the number of sessions containing at least one occurrence of each information need, and the number of conversations that each need occurred in among the 12 pairs.

We defined a conversation to be a single, complete request-response interaction between requesters and helpers. If turns of a conversation were about the same initial requester query,

Patterns	Description	# Sessions (out of 12)	# / Session	# Interviews (out of 12)	# / Interview
Background	Helpers wanted to know the requester’s experience level and background	11	1.5	7	0.7
Context	Helpers wanted to know what the high-level goals and context were	11	1.5	9	0.8
Sharing	Participants wanted a shared editor that lets helpers type code directly	11	2.1	11	1.1
Real-Time Response	Participants needed immediate responses (some preferred voice, others text)	12	N/A	10	0.9
Integrated System	Participants did not like switching windows, and would prefer a single system	9	N/A	7	0.8
Personalized Help	Requesters wanted help suited to their intent, e.g., specifying “teach me” when more explanation was desired	9	1.0	10	1.2

**Figure 3.4:** Common participant information needs that we observed during our human expert assistant study, with corresponding frequencies. Each of these needs would limit the success of a naive approach to providing remote assistance for software developers. “Number of Sessions” indicates the number of different trial sessions that each information need occurred in, while “Number of Conversations per Session” indicates the average number of times each need arose in a conversation (stemming from requester queries). “Number of Interviews” indicates the number of unique interviews in which the need was mentioned at least once. “Number of Mentions per Interview” is the average number of times that a participant mentioned the need in the post-trial interviews.

we counted them as a single conversation. If a requester asks a new question, it begins a new conversation if and only if there was new information elicited by the query itself. For example, if a requester asks a question, the helper replies with “Say that again?”, and the requester repeats their question, then this second query does not elicit new information, and thus would still be part of the same conversation. The same is true if the situation is reversed and the requester says “What?” to the helper’s response. However, if a helper first provides a response, and the requester replies by clarifying their query further, then we count the two questions as belonging to separate conversations. This rule separates interactions into focused pieces, and avoids imbalance due to participants’ varying styles of interaction.

Below, we provide evidence to support each of the claims in this section. In addition, we derived system feature implications that address the participant needs that we observed.

### 3.7.1 Experience

Seven participants, including both requesters and helpers, explicitly mentioned they would like to provide or have some kind of assessment of the requesters’ coding background in order to receive or provide help more efficiently. For example, if helpers knew that the requester is a jQuery beginner, then the helper could avoid simply telling them to write an AJAX call to make a request.

H1: *“If I knew where she was coming from, it made it more efficient, because I won’t have to ask do you know what this is, how to make this thing, or I could just say. . .”*

H1: *“... do you know console.log?”* R1: *“Yes”*

### 3.7.2 Context

Even though we designed the study such that the helpers do not know what the requester’s task is beforehand, requesters often rephrased part of the task to the helpers to provide context. We observed that helpers often helped refine the requester’s original questions and tried to get more context about the questions. Also, since we asked helpers not to be proactive and only use screen sharing when they need more information, they did not observe all the changes that the requesters made since the last time they saw the requesters’ shared screen. For example,

H1: *“She had it working correctly in terms of event title coming out on the web in the output, then later on when she was coding, and she was doing some quick copy and paste and stuff like that and some quick kinda changes and tiny bit of JavaScript, and I think she resize the, like different sizes in JSBin, or whatever it’s called, so I couldn’t see all JavaScript anymore, she’d changed something . . . , I couldn’t see what she did. . . I couldn’t see what she did later, it’s literally not like visible to my screen”*

R11: *“How can I make to fetch information line?”* H11: *“so what exactly are you requesting. . . what kind of information are you fetching?”* R11: *“it’s kind of like JSON file, I need to extract several information from the JSON file.”* H11: *“Right, do you have an URL or something?”*  
R11: *“Yeah”*

H5: *“If I could know what the problem is, the problem statement that he was solving for, so I probably would be able to help better.”*

### 3.7.3 Sharing

During the study, we asked requesters to share their screen. Both requesters and helpers mentioned that it would save time if helpers could type in the requesters’ editor or point to which line of code they were referring to. We observed that helpers often used the shared screen to tell requesters where to look and what to type. However, requesters were sometimes unable to follow helpers’ suggestions. This process took longer than expected, potentially because of the ambiguity of the helpers’ instructions (e.g., when multiple lines use the same terms mentioned), or the requesters were not familiar with the programming languages (e.g., what/where a callback

function is), or communication issues (e.g., English proficiency, or Skype signal quality). For example,

H1: “...just bring the cursor to the left into the after the curly brace...” R1: “after the curly brace, here?” H1: “yeah, and then put a comma, and put a string call JSON, and quotes” R1: “(typing)” H1: “no no the quotes...”

R3: “The code she sent can run on her side, but I’m not able to run it on my side. So if she could share it in some way, like do a comparison of the output, that will be great.”

### 3.7.4 Real-time Response

We provided voice, text, and sharing of the requester’s screen as communication channels during the study. Each of these channels delivers different information. Pairs mainly spoke to each other, even when helpers were instructing requesters what to type. In the post-task interview, two participants mentioned the real-time response is good for technical support based on their prior experience (H1, R3). The participants’ screen recordings also showed that when helpers were waiting for a request, they complete non-support tasks, such as coding their own project, reading news, etc.

However, more than half of participants were non-native English speakers, which led to voice communication issues. Thus, many requesters asked helpers to communicate through text. Furthermore, some requesters felt uncomfortable with voice and hesitated to ask questions as a result.

H6: “In general, I won’t write for the requester, but more like tell them [sic]....”

### 3.7.5 Integrated System

Participants used multiple systems during the study: Skype, code editor, browser, etc. We observed that participants were often annoyed when switching their focus back and forth between windows. For example, if a helper sent a code snippet to a requester via Skype, the requester would have to go to Skype, and copy/paste the code to their editor. On the other hand, when the helpers wanted to run the requesters’ code on their own machine, they had to copy the code from Skype and paste back into their editor. Beyond these issues, participants also had problems with connections and platform switching.

R8: “The hangout got disconnected. I was talking and developing, and no clue when it was

*not working. Also, I have to go back the window and check the text helper sent.”*

H3: *“Like every time I have to come back and check [Skype], it’s very tiny, maybe it’s because I shrink it, and then I have to open it up. . . if we could use JSBin, and then there is a chat or voice on the side bar, that will be faster, so it’s like I type and he can see and can talk. . . ”*

### **3.7.6 Personalized Help**

Often, we observed that requesters had to iterate on their original questions, even if the helper provided a related answer. Both requesters and helpers mentioned that they would like to provide or know the level of responses they want to receive from the helper or they should provide to requesters. This could be a high level answer, such as goal, or a detailed answer, like a function name. For example,

H1: *“Do you want to try to type something or you want to me to explain what to do?”*

R1: *“Please explain what to do.”*

R6: *“It would’ve been helpful if he could intervene to see if I’m doing something wrong ... that I couldn’t figure out.”*

## **3.8 Discussions and System Design Suggestions**

After analyzing the data from the hypothetical assistant study and the remote pair programming study, we found that both requesters and helpers expressed their concerns regarding our research question of how to provide or receive help more efficiently. We drew system design implications from these concerns and suggested six features to both helpers and requesters. We contacted our prior participants (11 of 24 replied) and interviewed them about the six feature suggestions we developed based on their experiences and concerns. We used prototypes to explain three of these features. We discuss the interviews and their design implications below.

### **3.8.1 Helper Page**

To make helpers efficiently find the questions they are capable of answering, we suggested creating a webpage that lists requesters’ unanswered questions. Here, we discuss two features related to this page.

## Question List

We prototyped this question list on a webpage that contains all the unanswered requests that the requesters made, code in the working file, requesters' highlighted code, a Cloud9 [89] link containing all the files in the repository that the requesters work on, and the requesters' Skype usernames. All of prior helpers claimed that this information is enough for them to determine if they are capable of answering a question. For some helpers, a well-written question in natural language was sufficient (H6, H10).

H4: *"If it's just whether you are capable of answering the question, I mean I would say like just have the question as specific as possible, with natural language."*

## Background Assessment

With the concerns about the requesters' programming experience, we suggest adding an assessment of requesters' programming abilities. We came up with four types of assessments and let the helpers choose their favorite: 1) the number of lines of code they have written in the past, 2) the frequency of their coding experiences, 3) the number of years they have been programming, and 4) a self-rating of Beginner/Intermediate/Expert.

Three out of five helpers chose assessment 2 (coding frequency) because they thought the other options could not be truly associated with the requesters' actual skill levels. In addition, they mentioned they would probably need more than the frequency: it would be ideal if they could also have the summary of code they have previously written. The other two helpers chose assessment 4 because they thought the other options could be biased. We also asked them whether they want the assessment result to be given in comparison to their own skill level or as an absolute measure. All preferred an absolute measure.

H6: *"I choose [assessment] 2. Because the total amount of code he wrote is a vague concept compared to how much his wrote in the most recent days and weeks."*

H10: *"[I choose assessment] 4. 1 is hard to tell. You can code a lot and don't understand anything. 2 is not a thing you can measure, someone's skill. 3 is not a measure."*

### 3.8.2 Suggested Support Features

On the requester side, we found that six participants directly mentioned the unfamiliarity they had with the JSBin editor. They would prefer to code in their own editor. Therefore, we



suggested to develop a package on a widely used editor. Below, we discuss four suggested features that are needed.

### **Code-Commenting Reminder**

We found that helpers often want to know the broader context of requesters' questions (their project and task goals), even when it was irrelevant to the request. Conversely, requesters also wanted helpers to be aware of their high-level task and plan. To address this challenge, we prototyped a feature in an editor that would automatically generate a descriptive comment reminder (e.g. *Please describe what this method does*) above important methods, such as API calls, which could make the code more contextual and readable. All six interviewed requesters found this to be a useful feature that reminds them to comment their code and aids helpers' understanding of their code.

R7: *"It would be a total nagging thing, so like you are risking annoying people, but that's exactly the thing that you need to nag, that you need to keep up with the commenting."*

Two participants suggested that this feature would be more useful if they could better understand how these methods were being used in the code and where they were called. This would allow helpers to better understand requesters' intentions. We also suggest making this feature easy to control; three participants mentioned that they would prefer to be able to quickly enable and disable this feature.

R12: *"[Often,] I forget to write my comments, it would be great to be reminded, it would save me a lot of time later. There is a small drawback when I don't have the time to do it, I don't want something to keep bugging me and telling me,... But if you can dismiss it, or if you can turn it off, it won't be annoying."*

### **Contextualized Explanations**

One setting in which requesters *do* prefer text responses is formatted code snippets. Helpers often want to include explanations with proposed changes to requesters' projects (either code or comments). Also, requesters may forget where and what they previously asked if the response time is long, thus, ideal responses would not only provide them the answers, but also where and what they asked.

H6: *"If I can directly see what he is typing, not just his screen. The screen resolution is not high, and I wish we could have something like Google Docs."*

To overcome this, we propose adding a contextualized explanation feature that checks all changes to the project code made by a helper, and displays the proposed edit along with an explanation generated by helpers. This may be similar in style to a modern word processor comment (e.g., in Google Docs, or Microsoft Word), which appears in a bubble to the side of the main text, with a visible anchor linking the explanation and the code. This not only helps requesters remain knowledgeable about the specifics of their code base, even when being supported by remote experts, but it also provides a chance to learn from the edits made, if desired. Three participants directly mentioned they would prefer to have such feature versus having the helpers type directly for them.

## **Text and Voice**

Participants could both type and speak to each other during the study, and they had different preferences in terms of asking or answering questions. Four out of six requesters preferred using text to communicate to make exchanging sample code easier. From the hypothetical assistant study, three out of five participants preferred text as well, and the rest mentioned it depends on the questions. We prototyped the feature that the requesters could ask a question in voice in the editor, and we suggest that the system should also allow participants to ask questions in text as well.

R11: *“I like a mix of both, but if only one, I prefer text because [a helper] can send the example. . . text will be very slow if I want to explain what I’m doing, voice can do this interaction very quick.”*

R7: *“[My preference is] probably voice, I found in general that whenever I ran into a problem, as soon as I can articulate the problem fully, I can find the answer to the problem”*

## **Teach/Show Me Button**

Ten participants expressed their wish that the helper could provide more personal help. They may want helpers to educate them such that they can fill in their knowledge gaps. Or they only want to speed their productivity up by simply completing the tasks. To be effective, this feature should be combined with a background assessment tool that allows helpers to better gauge requesters’ backgrounds. We suggest adding an option for requesters to indicate their expected response: if they want the helper to teach them about the question, or if they just want the helpers to show them how to do a given task. One requester mentioned that she usually

wants to integrate the example into her code so she prefers to understand the code first (R8).

R2: *“Also, if he can do this thing together, instead of he answers my general question, I prefer more detail question, I prefer he asked me something.”*

## **3.9 Chapter Conclusion**

This chapter examined a broad space of support systems and presented a first glimpse into the challenges and opportunities of on-demand expert support for software development tasks. Our studies provide insight into how and when developers choose to reach out for assistance, and the challenges of supporting developer needs via a remote expert. We have presented a set of features to address the primary needs that appeared in our study. We then validated the proposed utility of these features in a follow-up design iteration with participants from our trials. In summary, this work informs, and brings into focus, a previously under-studied paradigm of expert developer support.

# Chapter 4

## CodeOn: On-Demand Programming Assistance<sup>1</sup>

### 4.1 Introduction

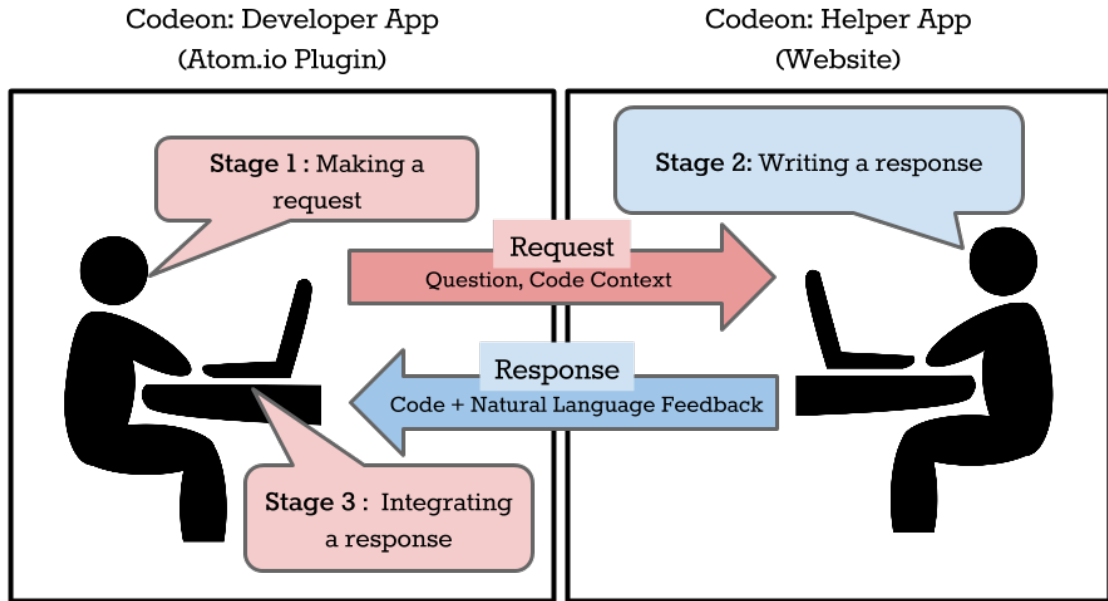
Driven by the findings of the previous chapter, we have implemented and evaluated this paradigm of on-demand expert support in CodeOn, a system that allows developers to request assistance as easily as they can through in-person one-on-one communication and tracks helpers' responses directly in the developer's IDE. As we will show, CodeOn makes remote collaboration more practical by reducing coordination costs while still enabling rich communication between developers and helpers. Unlike previous asynchronous collaboration solutions (such as code repositories), CodeOn is request-oriented: it allows developers to create sufficiently detailed requests and send them to other developers in a quick and effective process. Furthermore, CodeOn's asynchronous model is more scalable for multiple helpers than synchronous support tools because it allows multiple helpers to work alongside the developer. As our evaluation demonstrates, CodeOn supports new forms of parallel collaboration that make remote help-seeking more effective for developers.

### 4.2 Three studies to motivate CodeOn's design

CodeOn's design is based on the feedback we collected over the course of user studies of the three primary stages of help-request interactions: Stage 1) making a request, Stage 2) writing

---

<sup>1</sup>Portions of this chapter were adapted from [42]



**Figure 4.1:** Asynchronous interactions between developers and helpers can occur in three stages: making a request (S1), writing a response (S2), and integrating the response (S3). In CodeOn, developers use an IDE plug-in to make requests (S1) and integrate responses (S3), and helpers use a web-based IDE to view content and generate responses (S2).

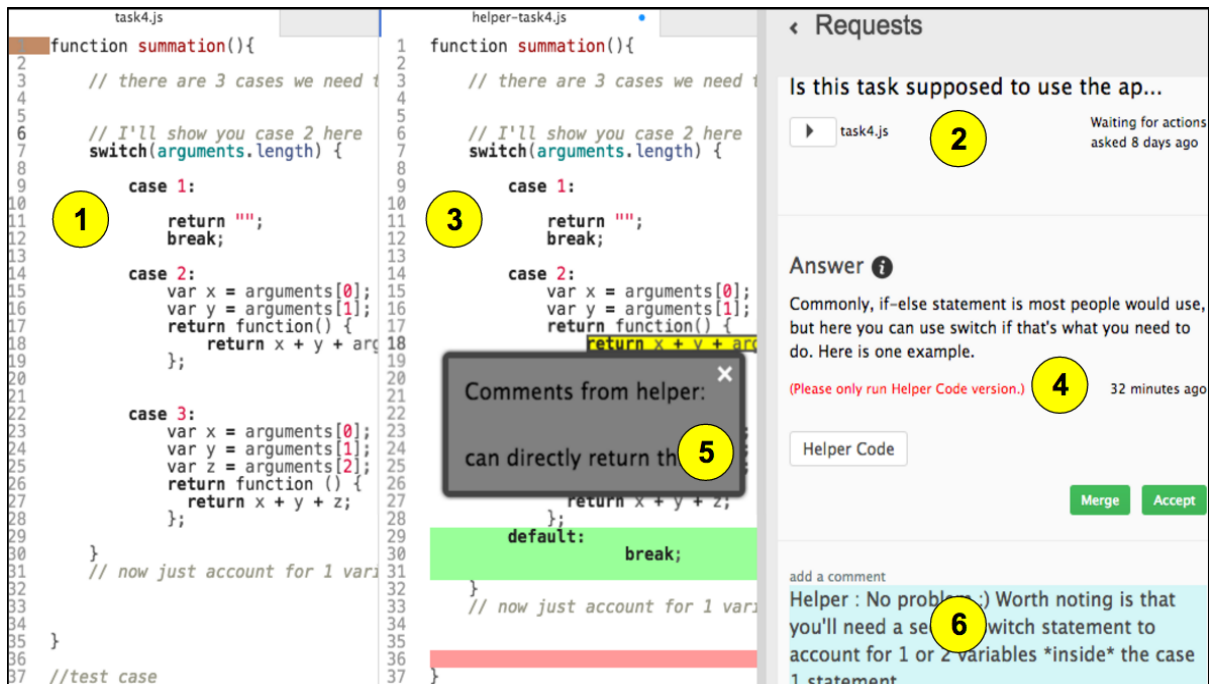
a response, and Stage 3) integrating the response (Figure 4.1). The design goal per stage is as follows: (*G1*): to simulate the in-person communication in seeking for help, (*G2*): to provide ways for a helper to associate responses with the working code context, and (*G3*): to make the code integration as effortless for developers as possible.

Separating the workflow into three stages enables better scalability by allowing a question to be presented to multiple workers and routed to a worker that has right expertise. These three studies aimed to help us better understand the trade-offs across different methods and features. To minimize the effects of varying prior expertise among participants, all preliminary studies used a synthesized programming language.

CodeOn’s developer interface is implemented as a plug-in for Atom.io—a widely used code

Modality	Voice	Text	Multiple Choice
Highlight	11.0 / 2.6	45.5 / 25.4	15.0 / 13.1
Click	14.0 / 3.7	37.5 / 23.1	27.8 / 18.0
None	21.5 / 8.4	33.9 / 17.4	25.1 / 9.5

**Table 4.1:** Time to make a request per condition (avg./s.d.). We found that spoken requests (“Voice”) where developers highlighted the relevant context were the fastest for developers to specify.



**Figure 4.2:** CodeOn interface where the requests and responses are on the right panel(2). Developer’s code(1) and helper’s code(3) are side by side for better comparison. Other responses includes explanation(4), annotation(5), and comments(6)

editor. It allows developers to make requests (S1) and visualize different formats of responses and integrate responses (S3) within Atom. For helpers, CodeOn provides a web-based IDE that allows them to see a list of developers’ requests and respond to them (S2).

## 4.2.1 Stage 1: Making a Request

### User Study: Asking a Good Question

With the primary goal of improving developers’ productivity, we designed this stage with two sub-goals in mind: 1) the speed of request making needs to be fast, and 2) the request needs to contain sufficient context to be understood. With these goals, we compared three modalities for describing requests: 1) speaking the request verbally (Voice), 2) typing the request (Text), and 3) selecting a request from a computed set of categories (Multiple Choice). We derived these categories from common query types observed in a previous study [40].

As prior work showed the importance of context in code request [40], we combined these modalities with three alternative methods for specifying the context of a given request: 1) selecting a region of content (Highlight), 2) pointing to one location in the content (Click), and 3) a control condition (No selector). Combining these request modalities and context selectors,

we formed a  $3 \times 3$  condition matrix for our experiment. We recruited 30 workers from Upwork to test the conditions and recorded the duration, content, and user activities of each request.

After removing the unanswerable requests (e.g. those that did not contain enough context), we found that, on average, voice requests were the fastest method for specifying requests, and text input was the slowest (Table 4.1). In the text condition, participants spent more time carefully crafting requests, whereas in the voice condition participants tended to speak more informally. The performance on the multiple choice option varied based on participants' familiarity with the options. However, we found in a subsequent evaluation of the questions' understandability that the multiple choice specifications were too vague to be understandable by helpers.

### **CodeOn Design: Voicing Requests**

As a result of our preliminary studies, we chose to use the speech modality for making requests. When developers make a request, CodeOn records their voice, synchronized with their interactions with the editor (typing, highlighting, file switching, and scrolling), which serve as the content selectors. As a request in voice is a dynamic signal, the content selector can also be dynamic so that one request can have an animation of not only the activity of content selection, but also some other informative actions such as typing or viewport changes. This way, a developer can speak and highlight code corresponding to the request, which can be replayed in the helper's interface. This simulates a pair-programming condition where a developer is asking a question from the person who is co-located by speaking and pointing to content on the screen. In addition, as a result of pilot studies with CodeOn, we also added a feature that allows developers to add an optional text title for each request for later reference.

## **4.2.2 Stage 2: Writing a response**

### **User Study: Response Modalities**

In this stage, we want to design features that allow helpers to provide different kinds of response format effectively and efficiently. We conducted a user study with participants responding to a simulated request. One response can have multiple parts and can be written in three different forms: 1) to select a segment of the code and to write an annotation that is associated with the highlighted segment (Code Annotation) 2) to write an explanation in a text box outside of the editor (Explanation), 3) to directly add and modify the code editor (Code Inline), and 4) the

Condition	Advantages	Disadvantages	Design Takeaways
Code Annotation	<ul style="list-style-type: none"> <li>• Preserves original code</li> <li>• Strong connection with the code context</li> </ul>	<ul style="list-style-type: none"> <li>• If the number and the length of annotations increase, they look littered and occlude the main code editor.</li> </ul>	<ul style="list-style-type: none"> <li>• Scalability needed while keeping high visibility</li> </ul>
Explanation	<ul style="list-style-type: none"> <li>• Preserves original code</li> <li>• Better suited for a long conceptual answer</li> </ul>	<ul style="list-style-type: none"> <li>• No connection between code and explanation</li> </ul>	<ul style="list-style-type: none"> <li>• Flexible and accessible</li> </ul>
Code Inline	<ul style="list-style-type: none"> <li>• Quick integration</li> </ul>	<ul style="list-style-type: none"> <li>• Possible merge conflicts</li> <li>• Low visibility</li> </ul>	<ul style="list-style-type: none"> <li>• High visibility necessary</li> </ul>

**Figure 4.3:** Advantages, disadvantages, and design takeaways for three response formats from developers and helpers' perspectives).

combination of all three types (All). We recruited 12 participants and recorded their performance on this task. Based on the common requests that participants had in Stage 1, we created three requests that each participant responded to. We measured the frequency of each answer type and conducted a post-study interview.

Our major high-level finding from this study is that participants' choice of response type depends on the request type, and each response type can support one or more types of response formats. Overall, we concluded that these three different forms complement each other, and the usage frequency of each type varied based on both the request type and the helper's preference. Table 4.3 details the trade-offs we found between different response formats in this study in terms effectiveness and efficiency.

## CodeOn Design: Response Generation

In order to allow helpers to easily view, understand, and respond to each request, the helpers' side of CodeOn is built as a web application where a helper can browse a list of developers' requests. Figure 4.4 illustrates the helpers' web interface. Once specific request is selected, the web application provides a programming environment that shows the files relevant to that request (files that were open in the developer's editor at the time of request generation). As mentioned earlier, a helper is able to not only play the audio that contains the question, but also see the developer's interactions with the Atom editor (e.g., text selection, scrolling, and content editing). Although the request might be involved only part of the original code base, all the scaffolding code are sent along with the request which makes the code executable.

Note that CodeOn does not support voice response because we want the system to be scalable such that not only can one worker supports multiple developers but also multiple workers can work on one question. Multiple voice responses will make the response review process time



consuming which violates our efficiency design goal. In our study, helpers used all three response types: code annotation, code inline, and explanation. In CodeOn, helpers can choose and/or combine different types of answers based on their preference and the question's characteristics.

### **4.2.3 Stage 3: Integrating a response**

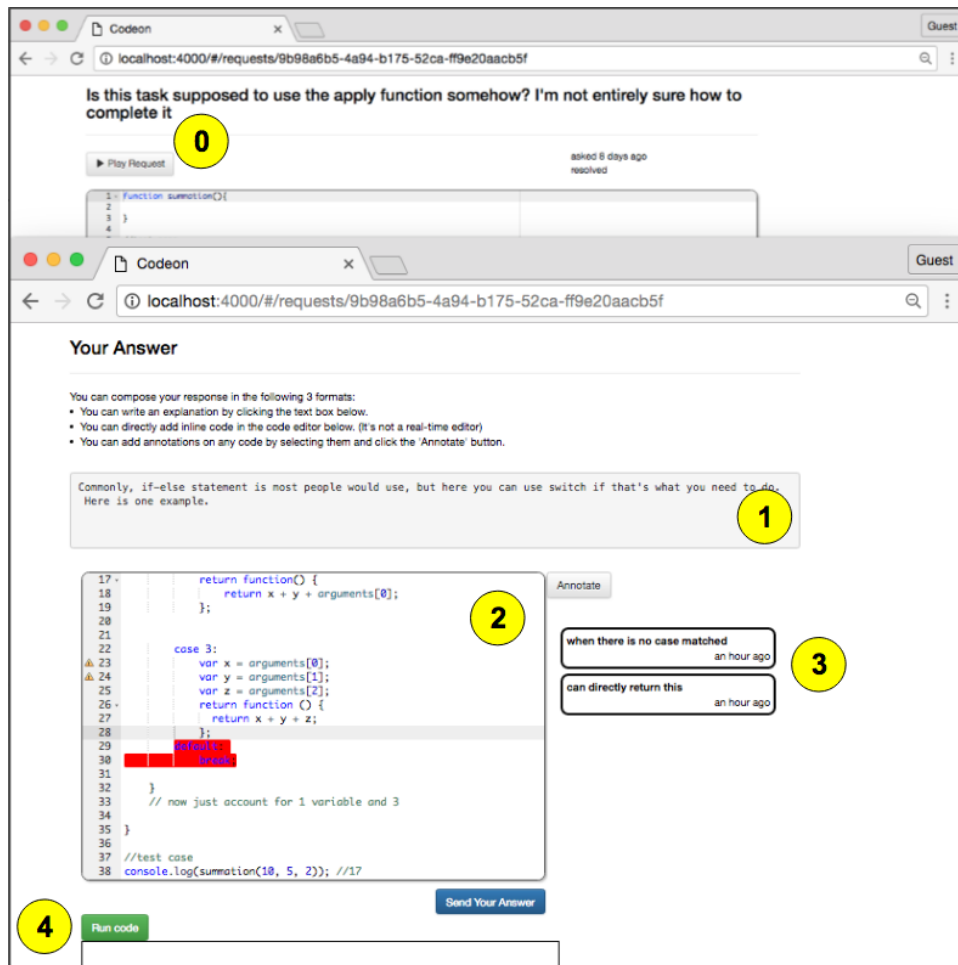
#### **User Study: Exploring Response Integration**

The last step of the workflow is to review helpers' responses and make changes in the original code. For this step, we want to make sure that the three answer forms (code annotation, explanation, and code inline) can be integrated accurately and quickly by developers. With these two sub-goals in mind, we ran a user study of how developers integrate the same responses presented in different formats. The experiment was composed of four conditions (one condition for each individual response form and one condition with all response forms) with three tasks. We measured the time and accuracy of code integration in each condition.

To make sure a response in different formats contains the equivalent information, we converted an answer in one form to another with specific rules. For example, when converting annotation and code inline to explanation, we specify line numbers to associate the content with specific line numbers of the code. We recruited eight participants with two for each condition, and we recorded their screen during the study, and conducted a post-task interview.

While we cannot generalize the findings due to the sparse number of participants, we can find that explanation took the longest for code integration and the code inline took the shortest. Similar to the conclusion from the second study, the post-interview indicated that there was no strong preference for one of the three types. Rather, participants expressed the trade-off between the types of answers and how each type of answer can be desirable and not desirable in some ways. For example, code annotation is desirable in a way that the textual content can have strong connection with a certain part of the code, complementing the explanation format (which makes it difficult to map the content to the code snippets). Additionally, participants preferred code annotation, as it preserves the helper's original code (and does not add new lines like code inline would). However, it is often seen as being appropriate only for short answers since it is overlaid code editor, meaning that it scales poorly as the number and/or length of annotations increase.

Similarly, the explanation format was desirable because it does not corrupt the original code, and it works well for both long and short responses. Compared to annotation and explanation,



**Figure 4.4:** The helper side of CodeOn is an interactive webpage that allows helpers to replay the request(0) and run the code(4). Helpers can respond to it with explanation(1), and inline code(2), and annotation(3).

inline code allowed participants to finish the task more quickly on average. While this is desirable to make the integration process efficient, another challenge is that a participant may miss the code inline added by a helper. This naturally led us to design measures to keep track of a helper's code inline, similar to the Code Diff application which will highlight the new code added. We summarized the advantages and disadvantages for each response format and drew design implications from the results that facilitate future system development (Table 4.3).

### CodeOn Design: Response View & Integration

CodeOn implements the response panel on the right side of the Atom.io editor. As there can be multiple requests and multiple formats per response, a scalable design is essential. The view consists of two hierarchical levels: the requests view and the response detail view (Fig. 4.2).

The request view has a list of requests where each menu shows the brief summary of the

request (title, associated file name, audio replay button) so the developer can keep track of multiple requests. Once a request is selected, the side panel shows the full information of the request and the most recently received response. In addition, if the response contains annotation or inline code, CodeOn will automatically split into a two-editor view with the developer's and the helper's code side by side. The region with annotation in the helper's code will be highlighted. When the 'Code Diff' button is clicked, CodeOn will display a color-coded difference between the developer's and the helper's code, similar to the 'diff' functionality in modern version control systems (e.g. Git).

Finally, one important goal of the response is to support efficient code integration. With the support of color-coded diff, integration of new code submitted by a helper to the original code can be done in one button click. In addition, for the common issue of the merge conflict in collaborative programming—when more than one person modifies the same content — CodeOn is designed to pull the most recent code (if there is any difference) to helper's side by default before helpers sending the responses. This is to reduce the workload of code merge for the developer. Lastly, to support conflict resolution and flexible integration, CodeOn generates clear annotated conflict markers for developers, allowing them to automatically merge the helper's code or to restore the original copy.

### 4.3 Iterative Design of the End-To-End System

We iteratively designed the complete CodeOn system based on the feedback from 19 developers who used CodeOn for a series of small programming tasks. In the initial CodeOn version, we found that developers could not 1) efficiently identify the code that responses corresponded to, 2) understand the differences between the original code and the helper’s edits, and 3) merge results from helpers into a consistent and functioning solution. The final version of CodeOn allows developers to: 1) better connect the content and the request by color mapping the line number and request panel, 2) better integrate helpers’ response by refining the code merge strategy, and 3) more easily view and compare to helpers’ responses with a side-by-side “Code Diff” tool.

### 4.4 Experiment

We conducted a laboratory study to better understand how CodeOn affects developers’ help-seeking behaviors.

CodeOn is built for any developers who seek programming support from remote experts. We recruited 12 students from authors’ university as developers with the requirement of at least six months JavaScript experience. We also hired three professional programmers as helpers from Upwork (upwork.com), an online freelancer platform, who self-reported multi-year JavaScript experience. The three helpers participated in multiple trials because we found little learning effect in our pilot study and to ensure that the helpers we used met our expertise criteria. We ran an hour and a half training session with each helper to familiarize them with the system and the study. We prepared two JavaScript task sets with each set containing four programming problems. These problems are independent on each other, and their answers cannot be easily found online. To ensure the two sets of tasks were as equally challenging as possible, yet conceptually different, we asked two professional JavaScript developers to balance the tasks. Every developer solved a series of JavaScript tasks in two conditions: a “control” condition and a “CodeOn” condition.

In the control condition, developers communicated with helpers via Skype (for real-time synchronous voice communication) and Codepen.io (for real-time synchronous code sharing). The control condition’s features are representative of the communication mechanisms that code mentoring sites use [90, 92]. In the CodeOn condition, Skype and Codepen.io were disabled, and the developer was instructed to use CodeOn to make requests. Both conditions allowed

developers to search for online materials. We collected audio and screen recordings during the study to capture the behaviors of the participants, and their responses to our follow-up questions. To minimize learning effects, we randomized the order of conditions (CodeOn, control) and the task sets (A, B).

We also instructed participants to finish the tasks as fast as they could by using any resource they were given (online materials and a remote helper), but did not explicitly suggest any strategies. Each study lasted one and a half hour, including training for developers (15 min), the two conditions (30 min per each), and the interview (15 min).

## 4.5 Hypotheses

We designed our study to evaluate four hypotheses:

$H_{\text{Performance}}$  : CodeOn can help developers to be more productive in development tasks than the control system could.

$H_{\text{SystemTime}}$  : Time that developers spend with CodeOn to ask for and get help is less than that in the control system.

$H_{\text{Interruptions}}$  : Developers get interrupted less often in CodeOn than in the control system.

$H_{\text{Parallelization}}$  : Developers can better parallelize their efforts in CodeOn than they can in the control system.

## 4.6 Overall Performance

The productivity of each condition was measured by counting the number of completed tasks (out of four tasks per condition given 30 minutes cutoff time). Figure 4.5 shows that the average number of completed tasks within the given time in the CodeOn condition is significantly more than it is in the control condition (two-tailed paired-samples Student's T-Test,  $p = 0.03$ ), which supports our hypothesis  $H_{\text{Performance}}$ . To understand *why* developers were more effective with CodeOn, we further analyzed our user data, as we will describe in the following sections.



**Figure 4.5:** Overall result: The # of completed tasks is significantly more in CodeOn condition(avg./s.d.).

## 4.7 Individual Task Performance

To understand the advantages of CodeOn, we unpacked our data to investigate participants’ performance on each task. As Table 4.2 shows, participants spent less time in completing tasks on average when using CodeOn condition, although the difference is not significant. While CodeOn may help developers to complete tasks quicker than the control model, the difference is not significant enough to be the sole factor in CodeOn’s result. Meanwhile developers may waste more time on a task when they get stuck with it in the control condition. The time spent per incomplete task also support this conjecture. Especially if we exclude the incomplete tasks that were stopped by the researchers for 30 minutes (“tail condition”), we can observe a 23% time increase in the control group. As the time spent on the incomplete tasks were determined by an external factor (the time constraint), not by the developer’s intention, we believe this

	CodeOn	Control	Time Increase(%)
Time spent per completed task	10.57	11.32	7.1%
Time spent per incomplete task	8.49	9.15	7.8%
Time spent per incomplete task in non-tail condition	6.84	8.47	23.7%

**Table 4.2:** The average time (in minutes) spent per task. Participants spent longer in the control condition, on both completed and incomplete tasks. Tasks in tail condition is the task that are stopped by the researchers by the time constraints (30 min). Note that, by definition, there cannot be complete task in the tail condition.

name	CodeOn	Control	p-value
Avg. # of requests per completed task	1.71(1.41)	2.18(1.54)	0.45
system active time(sec) per completed task	165.8(106.3)	344.4(249.5)	0.05

**Table 4.3:** The # of requests made per completed task is not significantly different. The average system active time per completed task in CodeOn is longer than in the control condition (avg./s.d.).

measure better reflects the time spent on incomplete tasks for the comparison purpose. While we cannot calculate the statistical significance for these three measures as samples in each condition are part of the entire data set (e.g. complete tasks in CodeOn are different from the complete tasks in the control condition), the results indicate that the improvement in overall productivity potentially comes from wasting less time when the developer cannot solve the problem in CodeOn. We hypothesize ( $H_{\text{Parallelization}}$ ) that the asynchronous nature of CodeOn workflow encourages developers to hand off their work to a helper and to move on to the next task, whereas, in the pair-programming session, two developers typically work on the same task at a time. In the next section, we evaluate if developers parallelize work efforts during the experiments.

As we do not find strong evidence of developers completing tasks faster in CodeOn, we further analyze how actively developers utilize the assistance system when they were able to complete tasks so as to give an account of the increase in overall performance. The average number of requests and *system active time* per complete task is reported in Table 4.3. System active time is the time that a developer spent on the assistance system (CodeOn or the control system) to make requests to a helper and to receive assistance from the helper. System active time thus includes any time that would not have been needed if there was no helper, for example, watching the helper programming (in CodePen), creating a request, reviewing responses from the helper, or interacting with the helper (via Skype, CodePen or CodeOn). The result shows we cannot see a significant difference in the number of requests made per complete task ( $p = 0.45$ ). The system active time per completed task in CodeOn, on the other hand, is less than the one in the control condition ( $p = 0.05$ ), which shows a tendency to significance for our hypothesis  $H_{\text{SystemTime}}$ . Based on our self-assessment from the video annotation process, we notice that the cost of extra time in the control condition may come from the nature of synchronous communication between two ends. For example, a remote pair-programming session may cost

name	CodeOn	Control
Avg. # of alerts	6.1(3.0)	1.9(2.2)
Avg. # of interruptions	2.5(1.6)	1.9(2.2)
Ave. of interruption/alert	0.48(0.3)	1.0(0)

**Table 4.4:** # of interruptions, alerts, and average of individual ratio of interruption/alert(Avg. / S.D.).

additional time coming from social norms, real-time typing process, additional out-of-context questions (or feedback) [48] that may not contribute to the overall performance and does not exist in the CodeOn model. This aligns with our belief that CodeOn is more efficient in seeking for and receiving help from a remote assistant. The efficiency in CodeOn can potentially cause an overall increase in the performance by expediting completion time or giving the developers more time to complete.

## 4.8 Interruptions and Parallelization

Studies have shown that interruptions can be costly to programmers [94]. As CodeOn follows the asynchronous collaboration model, we analyze the occurrences of a helper interrupting a developer and evaluate if it has any advantage of being less disruptive to the developers. Annotating the screen recordings of each experiment, we count the number of alerts and interruptions. An *alert* is a message from a helper that initiates a conversation, which gets an attention from the developer or notifies the developer that a response/comment is received. Receiving an alert does not necessarily mean that the developer needs to take action immediately or is interrupted. For example, in CodeOn, a developer can see the notification of a helper’s response and review the response later, once the work being carried out is done, or, in the control condition, the developer can ask the helper to wait a little while. In addition, the task that the developer was currently working on may be directly relevant to what the helper responded so that the interrupted task not needed to be resumed. We say an alert causes an *interruption* if the two following conditions are satisfied: i) the alert makes the developer immediately stop what they are working on in order to review or respond to the helper’s message, and ii) the stopped task needed to be resumed later. Table 4.4 shows the absolute numbers of both alerts and interruptions are greater in the CodeOn condition. This is because in CodeOn, one comment is counted as an alert, whereas in the control condition, the developer and the helper constantly communicate so that they have a smaller chance to be interrupted as they are working together.



However, when a helper alerts a developer in the conference call, the developer has to stop the current task 100% of the time. In the meantime, in CodeOn, they were interrupted (immediately respond to the helper and later resumed the task) only half of the time (48%), and otherwise they could keep working on their task until the point that they finish the current activity (e.g. finishing the line that was being written, finishing reading online materials that were being read). Even when developers were interrupted in CodeOn, we observed that most of the interruptions did not require a significant context switch in the developer's mental model as the interrupted task was relevant to the response from the helper. We did not choose to evaluate this as it can be subjective. If we look further detail for individual, 5 out of 12 developers chose to wait to review responses and, on average, they spent 18.1 seconds to finish the ongoing activity. Potentially, this tendency can scale once the system is deployed and is constantly used by developers. Indeed, using CodeOn, developers can have better control over their workflow by having a smaller number of interruptions ( $H_{\text{Interruptions}}$ ) whereas, in synchronous collaboration, the workflow will be determined by the pair otherwise the developer will be interrupted.

As briefly mentioned, another benefit of asynchronous collaboration can come from a developer parallelizing the task by handing off subtasks to helpers. To confirm the possible benefit in CodeOn, we annotated the video to see if developers parallelize their work while waiting for responses from helpers. We present the number of parallelization and the time that the developers parallelize their tasks in Table 4.5. Table 4.5 presents that developers parallelized their work 2.1 times on average when they hand off their work to the helper ( $mean = 2.1$ ) in CodeOn condition, whereas in the control condition this behavior occurred close to zero ( $mean = 0.3$ ). In addition, their time spent on parallelization is much longer in the CodeOn condition. Furthermore, the two developers with parallelization behavior in control condition were instantly interrupted ( $mean = 1.9s$ ) by helpers when they attempted to work on different tasks. We found that 11 out of 12 developers parallelized their work when using CodeOn, but only two when using the control system. Thus we can assume the parallelization is natural in the setting of CodeOn. The result supports our hypothesis  $H_{\text{Parallelization}}$  that CodeOn supports the distributed workflow. This potentially account for the improvement performance. The evidence and analysis above provide us with insights on the overall performance of two systems. Next, we review developers' feedback and screen recording to facilitate the qualitative analysis.

name	CodeOn	Control
Avg. # of parallelization per developer	2.1(1.2)	0.3(0.6)
Total time(s) of parallelization	281.9(243.5)	2.4(5.7)
Avg. time(s) of parallelization per occurrence	114.2(80.5)	1.9(4.8)

**Table 4.5:** Time spent and the number of parallelization behavior in two conditions (Average / S.D.).

## 4.9 Interview and Developer Feedback

### 4.9.1 Parallelization

Nearly every developer (11/12) parallelized their efforts. We discovered two patterns of parallelization behavior from both post-interviews as well as our observations. After sending a request or comment, developers would either 1) review a different task, or 2) work on another part of the same task. The first pattern is more common, and some developers used it directly after they read the problem.

*“.. after I read task 4, and I found the length of the description is very short and I think it’d be very easy to just type the task very quickly and make it clear of what to do (for the helper), so I just gave the helper this task. (P5)”*

The second pattern often happened in those tasks with multiple requirements. Developers would divide the task into a few subtasks and distribute some to helpers. For example, one task asks to remove the duplicates of an array and then sort it. One developer (P4) asked his helper to write a function to remove the duplicates. While waiting for a response, he started to code the sort method. Another developer (P9) moved on to a search task after making requests about writing a method and code debugging.

*“I was able to kinda break down the tasks into subtasks, and kind of, things I can ask him to help with, and things I can work myself. (P9)”*

In general, we observed that developers consistently showed a tendency to parallelize their work, regardless of the condition. However, we found that CodeOn allows them to accomplish the distributed workflow.

### 4.9.2 Interruption

The post-interview also supports our hypothesis  $H_{\text{Interruptions}}$  by having five participants directly mention the interruption issues in the control system (none in CodeOn). There are two types of

interruptions we noticed. One is direct interruption which we defined in the previous section, and the other is more subtle distraction coming from the conference call itself. For example,

*“I can just hear him in the background, it’s kind of, not intimidating, but like, make me feel like I had to ask questions, even though I wanted to do stuffs on my own. (P11)”*

*“When using Skype, he kept asking me about clarifying things that I asked him, I couldn’t do anything at the same time, like I had to pay my attention to what he’s asking and make sure that whatever I’m asking him, he understood properly. (P9)”*

Three developers in the control condition, although working on other tasks while waiting for helpers, were interrupted by their helpers (e.g. asking for confirmation, requesting to check for answers). For example, one developer asked the helper to “just do the (first) task for me” (P4) in the beginning of the control session. Then, the developer moved on to the second task while the helper working on the first task. The helper started to type code in the editor and to explain what s/he was typing at the same time, which distracted the developer. The helper regularly asked for confirmation such as “you see this?”, which force the developer to switch applications back and forth to interact with the helper until they eventually decided to solve this problem together.

## **4.10 Discussion**

### **4.10.1 CodeOn User Interface**

Almost all the developers (11/12) in the experiment gave positive feedback on CodeOn’s user interface, that supports our design decisions retrieved from the series of user studies.

For example, participants mentioned that making requests by using multimodal interaction (voice + code context) allows them to “give context easier” (P4). Furthermore, participants were generally in favor of the way in which CodeOn integrates code-based responses. For example, the pop-up notification and the alert sound coming with the new message helped developers to be notified more quickly.

*“..the beep sound, I liked it. Like the alert that the notification has been sent. Because when I asked some questions, and then concentrated on other things, it might be easy to forgot the previous questions .. the beep sound reminded me about the previous questions .. (P12)”*

*“..the notifications that it gave me are very good... comparing to only have text, add sound can help me to... when i look at the left i can still know what happens on the right.(P5)”*

CodeOn also prevents developers from “missing something that a helper wrote”, and helps

them better understand the code by allowing them to “compare two code files simultaneously” (P7). Two developers mentioned that they could not follow where the helper was typing in the control condition because, unlike CodeOn, the interactions cannot be replayed nor easily recorded.

*“In Codepen, the helper is changing in another window (other than Atom) that I have no idea what he did. In Skype, if I have a question I just say it but there is no history. (P3)”*

#### **4.10.2 Effects of Social Norms**

Previous research shows that lower social burden on asynchronous communication activity than synchronous [8]. We also found that developers in the control condition expressed the challenge in real-time communication: phrasing the requests, or explaining the code. With the study setup of having a remote helper available in real-time via a conference call, four developers addressed that they were less comfortable and felt more pressure because they felt they “have to ask something” (P11), and less comfortable when having someone just “sitting there” (P11, P4). The rest of them felt little pressure and relied on helpers more to solve the problem. On the other hand, no one expressed similar concerns for CodeOn. CodeOn offers a more independent environment with little social pressure and the full control over code and the assistance pipeline.

*“In Skype, but I also felt like, not that he’s interrupting me, but like I can just hear him in the background, it’s kind of, not intimidating, but like, make me feel like I had to ask questions, even though I wanted to do stuff on my own. (P11)”*

#### **4.10.3 Potential of CodeOn**

One of the limitations of our study is that there are only four tasks, which limits CodeOn’s potential to support parallelization better and may lead to a larger difference in productivity. For example, there is one developer finished all four tasks in the CodeOn condition within 30 minutes. Also, we found that the developers, who only have one task left before the session ended, cannot parallelize their effort(as they could before) because there are no other tasks left. Instead, after making a request for one task, they would continue working on that same task which creates redundancy.

In contrast, many developers expressed concerns on the control system for the synchronous nature of the collaboration. For example, as two developers are sharing an editor, the editor can

be a limited resource that blocks a developer's interaction. Participants expressed that they felt limited for various reasons: being "stuck watching" (P12) the helper's typing, being distracted with the other "hear him in the background" (P11), or "delete my code and directly add his code" (P7). Especially given the social barrier in which a developer is put to work with a remote stranger, the pair programming model revealed some challenges for developers in engaging with the helpers in a short time.

#### **4.10.4 Generalizing CodeOn's Approach**

Our target audiences are programmers who need support that can be more efficiently provided by remote expert developers than existing methods. This remote assistance model can be useful in many contexts including education and distributed teams. We focus on developers' communication, which is important in all of these use cases, regardless of team size or incentive. The three stages of CodeOn system we have discussed before fit into a more general model that CodeOn advances. Specifically, tools for generating sub-tasks in the current context of work (S1), making it easy for helpers to 'rehydrate' that context (S2), and providing tools for quickly and effectively integrating contributions (S3), can be used across various domains, for example, using software made for professionals (e.g. Photoshop or Final Cut Pro) Exploring how to effectively recreate this approach in other settings is future work beyond the scope of this paper.

### **4.11 Chapter Conclusion**

In this chapter, we described CodeOn, an in-IDE tool that allows software developers to receive asynchronous on-demand assistance from remote programmers with minimal effort. Our results showed that developers using CodeOn are able to complete nearly twice as many tasks as they could using state-of-the-art synchronous video- and code-sharing tools by reducing the coordination costs of seeking assistance from other developers. CodeOn is an early step towards consistently available, crowd-powered developer assistance. It instantiates the paradigm that we proposed earlier in the thesis. In the next chapter, I will discuss a study that explores on-demand collaboration in embedded system development.

# Chapter 5

## CoCapture: Facilitating Communication about Dynamic Interactive UI Behaviors<sup>1</sup>

While CodeOn effectively supports requests related to code syntax, many other requests are related to non-code or even non-textual information. Requests related to UI behavior, for example, typically require UI designers or developers to include visual information, such as a mockup, as a shared context. In this chapter, I discuss the challenges of communicating dynamic interactive UI behaviors on existing websites and present CoCapture, a system that addresses this issue by making it easier for developers to create and annotate visual references that effectively communicate about UI issues or proposed changes on existing web interfaces.

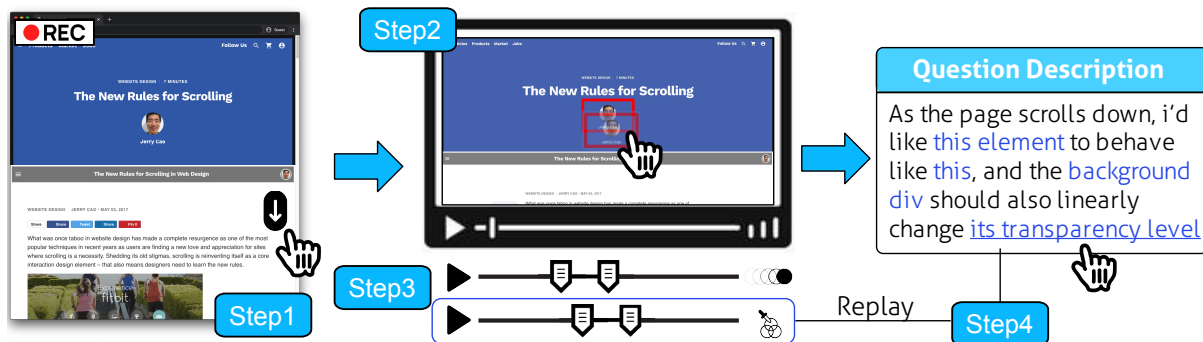
### 5.1 Introduction

Mockups are widely recognized in HCI as invaluable tools for communicating and evaluating design ideas. A mockup can help ground descriptions of UI functionality and can serve as a “boundary object” that allows designers to communicate with developers and other stakeholders. Mockups are useful *throughout* the UI development lifecycle, from exploration to refinement. However, most tools for mockup creation are built for the earlier (exploratory) stages of UI development.

There are several challenges when creating mockups as communication tools in the later stages of UI development—for example, to propose changes to a UI that already works or to describe a desired behavior in a UI that contains an error. First, most tools for creating

---

<sup>1</sup>Portions of this chapter were adapted from [38]



**Figure 5.1:** The Workflow of CoCapture. There are four steps of using CoCapture to communicate new UI behavior mockups on an existing website. **(Step 1)** Users first capture existing interface behaviors (base scene) by interacting with the website (scrolling), and CoCapture will automatically capture the DOM changes. **(Step 2)** In CoCapture’s main panel, users can add new behaviors on top of the base scene by demonstration; that is, by directly manipulating any elements (e.g., drag and drop the red element in the replay and see immediate changes). **(Step 3)** Users can remix (post-edit, e.g., change duration) added behaviors to finalize the mockup. **(Step 4)**. Users can refer to the DOM elements or added behaviors in the textual description using hypertext.

mockups cannot import assets or behaviors from existing UIs, and it can be tedious to replicate the intricate details of a working UI in a mockup. Second, it can be difficult to communicate how an existing UI should change because it is not easy to point out the difference between the existing behavior and the mockup’s behavior—particularly when the change is a nuanced or dynamic behavior [144, 157, 149]. Third, mockups that propose changes to existing behaviors often need to be *mixed-fidelity* [134]—with high-fidelity representations of existing components and low-fidelity renderings of proposed changes—but few mockup tools support this. These limitations led to our research question: *How can we make communicating about changes to existing UIs easier and more effective?*

In this paper, we introduce CoCapture, an interactive system that enables users, like UI designers, to **easily create** and then **accurately describe** UI behavior mockups. These mockups could be changes the users want to propose or questions they want to ask about an aspect of the existing UI. With CoCapture, users first record the existing UI behavior by *demonstrating* an example interaction on the existing UI (Fig. 5.1, Step 1). Building on this scene, users can further create DOM element level dynamic behaviors via *demonstrations* (Fig. 5.1, Step 2) and *remix* these demonstrations as a first-class animation object (Fig. 5.1, Step 3) through direct manipulation and low-fidelity sketching. To help accurately specify the visual changes, users can write natural language descriptions in CoCapture that contain hypertext references to specific aspects of the mockups (e.g., specific DOM elements, new animated effects) (Fig. 5.1, Step 4).

We conducted two within-subject studies to evaluate the communication effectiveness of multiple aspects of CoCapture: the effort of creating visual context and the accuracy and clarity of the description. Our results show that compared to traditional sketching and communication tools, the requester participants using CoCapture spent less than a third of the time on text writing, and their descriptions of UI behavior were significantly more accurate. Additionally, the helper participants reported that the descriptions in CoCapture were more accurate, concrete, vivid, and easier to follow.

The **key contribution** of CoCapture is a novel interactive method that combines the DOM-element-based recording technique with a demonstrate-remix-replay approach. This makes it easier to prototype on pre-built UIs and to describe user needs regarding dynamic UI behaviors more accurately than possible with existing approaches. With CoCapture, users can effortlessly explore different possible designs, capture fleeting ideas, and communicate with others about behavior ideas on existing interfaces. Specifically, our contribution includes:

## 5.2 Need-finding Studies

We conducted two studies to better understand designers' challenges and needs when communicating about UI behaviors.

### 5.2.1 Stack Overflow Analysis

We first conducted an analysis to identify the categories of questions related to UI behaviors that were most frequently asked on Stack Overflow (SO), a well-established Q&A platform in the programming community. We analyzed the 200 most viewed questions that were tagged with JavaScript (JS) and CSS. We used these two tags because they are primarily used for manipulating UI elements (JS) and presenting them (CSS). We read through all the posts, counted other included tags, and documented the use of visual references in the posts. We found that 41 posts (20.5%) included tags that were CSS properties (e.g., height, position) or were related to interface element changes (e.g., CSS transitions, sticky menus). We also examined the visual references included in each post and found that 34 posts (17%) included links to a live example of their problem,<sup>2</sup> 18 posts (9%) included screenshots, and 5 posts (2.5%) included sketches. This helped us determine which of the most frequently asked question categories also required

---

<sup>2</sup><https://stackoverflow.com/a/{24414642|5445491|17722497}>



visual information. We used this information to guide our system and study design.

## 5.2.2 Needs and Challenges

Our second study examined how well existing tools can support communication about common UI behavior issues on existing web interfaces. We recruited eight participants with at least one year of UI design and development experience (3 female, 5 male) from the authors' university. Among them, half acted as requesters to ask three questions using Scrimba [174], a state-of-the-art tool that allows its users to simulate in-person communication by recording voice narration and editor activity (e.g., typing, highlighting). The three questions represent the most popular categories we found in our Stack Overflow analysis: a responsive UI task,<sup>3</sup> a platform game,<sup>4</sup> and an animated effect applied to an object.<sup>5</sup> The remaining participants acted as helpers, reviewing the clarity of the requesters' three questions by comparing their understanding of the requests to the ground truth desired output (presented as a video demonstration). All helpers were teaching assistants for a UI development course. After the tasks, we held a follow-up interview with all participants in order to help inform the design of CoCapture.

Instead of describing the desired change in the interactive behavior via text, which would result in biases [155], we gave requesters the code (i.e., HTML, JS, CSS), the existing UI, and the desired UI for each task. We also pointed out, with visual annotation (e.g., cursor pointing) and determiners (e.g., "this part"), the differences between the behaviors of the existing and desired UIs without offering specific descriptions. Requesters could access this information throughout the session but were not allowed to use any materials from the desired UI artifact (e.g., no screenshot of the desired UI). This simulates a scenario in which a UI designer knows what the desired change is but does not possess a visual representation of the new behavior.

The lead author conducted all of the studies in our research lab. Each session lasted approximately 45 minutes. Sessions were recorded and transcribed. The lead author went through the transcripts and coded them using an open coding approach [33], which included discussions with the research team. We report our findings for requesters [R] and helpers [H] below.

**Challenge 1 (C1): [R] Visual context is necessary but difficult to describe on existing interfaces.** When asked about their experience describing the UI questions on the given artifacts,

---

<sup>3</sup><https://tinyurl.com/ydev4uwr>

<sup>4</sup><https://github.com/starzonmyarmz/js13k-2018>

<sup>5</sup><https://semantic-ui.com/modules/transition.html>

all requesters noted the challenge of providing visual information in the request: *“I wish there was an easier way to describe the difference between what’s there [existing UI] and what’s needed to be there [desired UI]”* (R1). When asked about their experience with UI behavior questions on Stack Overflow, requesters also expressed the necessity of including the contextual information of visual changes as part of the requests. *“I’d prefer to create a video to demonstrate the whole changing process”* (R4). When asked about their needs when creating the requests, R1 suggested having a better way of organizing the information. This indicates the importance of providing visual information changes in a request, as well as a need for easier methods of adding visual information changes to existing interfaces.

**Challenge 2 (C2): [H] Natural language descriptions may lack the necessary details.**

Four helpers commented that some of the questions were *“too vague”* (H1), *“broad”* (H2, H3), *“unclear”* (H4), or *“hard to understand”* (H2, H3). For example, H2 said, *“I don’t understand when they say their HTML pages [are] not responsive, because it is responding to the way they’re resizing.”* To respond to these “vague” questions, two helpers said they had to provide suggestions based on their best understanding of the questions (H1, H2), while the other preferred to follow up with clarifying questions to make the expectations more explicit (H3, H4). This suggests that the current tools might not provide sufficient support for beginners to easily phrase their questions and receive the correct assistance.

**Challenge 3 (C3): [R, H] Voice-based video requests can be tedious to make and navigate.** Both requesters and helpers reported that they were more used to tools with text (e.g., email and text-heavy images like annotated screenshots) for asynchronous communication, and they found voice-based requests difficult to use. Requesters felt creating voice annotation was *“time-consuming for people who have to think long [about] what they have to say”* (R1), *“hard to synchronize everything together”* (R2), and *“hard to edit later and so that kinda make me more nervous to do”* (R3). The other requester (R4) felt voice recordings were an easy way to annotate visual requests. Helpers commented that the voice requests were *“helpful”* (H2), though they wished the recordings were *“more slowed down”* (H1) and *“easier to find information like searching through text”* (H3). This feedback suggests that text-based request can be more approachable for requesters to make and for helpers to comprehend information. In addition, referring to dynamic behaviors and visual elements remains a challenge, one we aim to address with our system.

**Need (N): [H] Seeing the existing and desired behaviors would facilitate helpers’ work.**

When asked about the necessary information for understanding a UI behavior question, all helpers chose the existing output and desired output. Half of the helpers felt the related code would not be necessary, as *“there’re so many ways of doing something. As long as they’re able to understand what the results should be, I can figure out what the code is supposed to do”* (H2). This indicates that both the existing and desired behaviors should be included in a UI behavior question. After being shown the ground truth of the desired behaviors, three helpers found that they had misunderstood at least one question, noting that *“I thought it was going to be this whole block that was moving together”* (H2) or that *“I was actually thinking that this bar here would stay on top above this”* (H3). Helpers suggested creating a video that scrolled through all of the different sketches and screenshots, a concept that aligns with suggestions from prior work [157]. These results indicated the need for a tool that highlights the moving elements that respond to the UI behaviors.

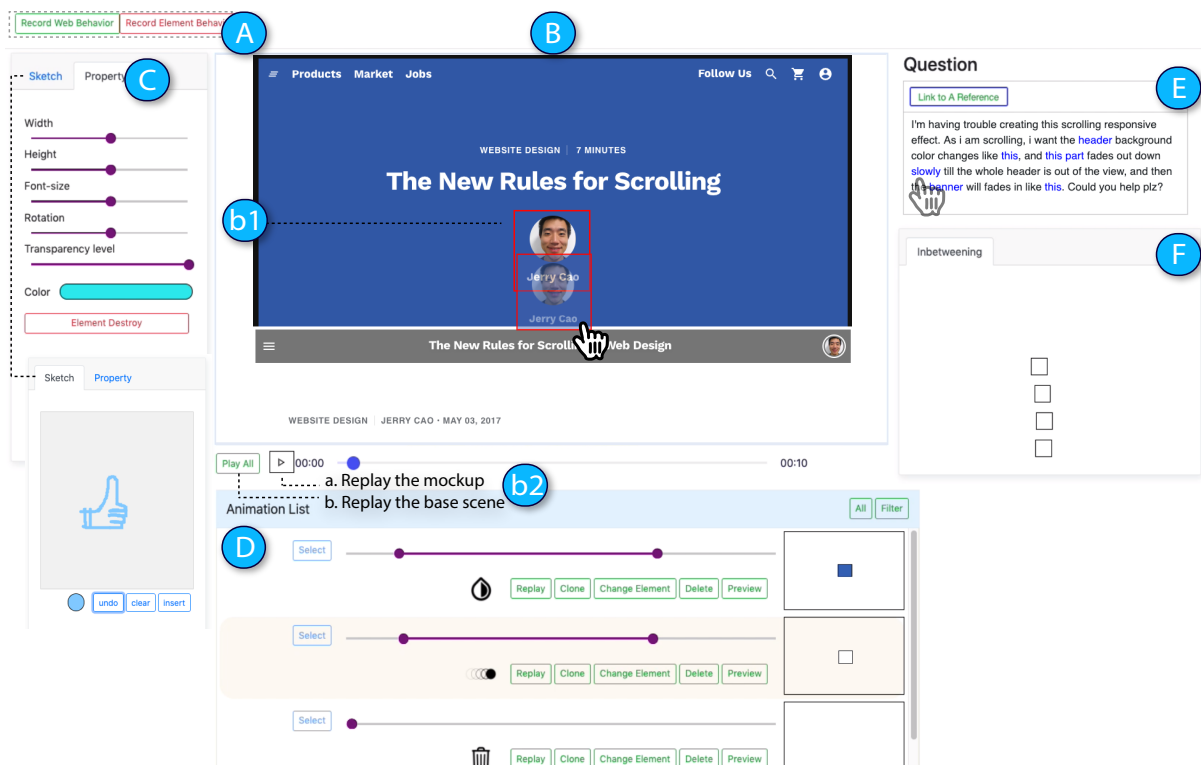
### 5.2.3 Design Goals

Driven by these findings (C1-C3, N), we came up with three goals (DG1-DG3) to guide us in designing CoCapture. Fulfilling these goals would allow requesters to easily prototype behaviors on pre-built interfaces and effectively communicate the new behaviors to helpers.

- **DG1: Users quickly create visual changes (C1, N):** Requesters need to quickly and accurately create their desired UI mockups on existing interfaces and include both the existing and desired behaviors in communications.
- **DG2: Requesters accurately describe behavioral information (C2, N):** Requesters need to easily and accurately describe the desired behaviors of their created mockups, not just the desired appearance.
- **DG3: Helpers understand requesters’ needs with ease and clarity (C2, C3):** Helpers need to easily and clearly understand the existing and desired UI behaviors, with each behavior being accurately specified.

## 5.3 CoCapture Design and Implementation

Guided by the design goals, we developed CoCapture, a Chrome extension that allows users to easily create mockups on an existing website through direct manipulation and to reference pieces of the resulting mockups in their description using hypertext. In this section, we first



**Figure 5.2:** CoCapture’s main panel (after Step 1 in Fig. 5.1). To create an animation, a user first clicks an existing DOM element (b1) (and can select multiple by holding the `Shift` key) from the base scene (B). Once recording is started (A), the user can demonstrate the desired behaviors by changing the elements’ properties (C) or directly dragging them in the base scene. Once the recording is finished, the demonstration will be appended to the animation list (D) with a set of meta operations including options to adjust the starting/ending time, replay, and preview. The user can write their question (E) and refer to the animations or DOM elements in the scene by selecting portions of text and clicking the “Link to a Reference” button. Text with references, or hypertext, is in blue with a click affordance displaying the relevant context (i.e., highlighting elements (b1) replaying animations (D)). The process of the animation (inbetweening) is also displayed (F). The user can also filter the animation list (D) to only display the animations related to the selected elements.

illustrate the experience of using CoCapture with a sample usage scenario that embodies many of the use cases identified in our formative studies. We then detail the design of CoCapture.

### 5.3.1 The CoCapture User Experience

Jerry, a junior professional web interface designer, is in the middle of prototyping a website that has most of its content ready (Fig. 5.3). Now he wants to extend the website by adding some interactive behaviors to respond to a user scrolling event (Fig. 5.4).<sup>6</sup> He decides to use CoCapture to create a mockup of his vision and then ask his peers for feedback. While on his

<sup>6</sup>A similar fade-out example: <https://html5up.net/massively>

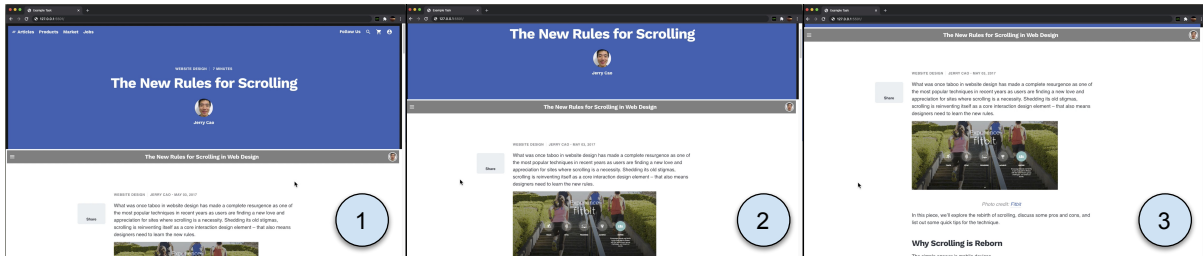


Figure 5.3: Three screenshots of Jerry's current user interface at different scrolling percentage.

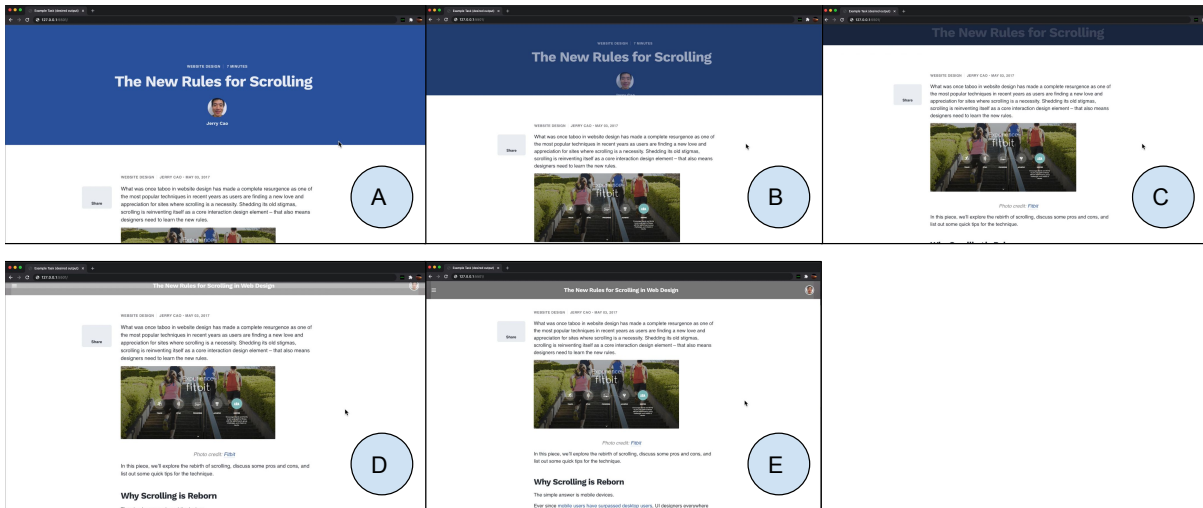
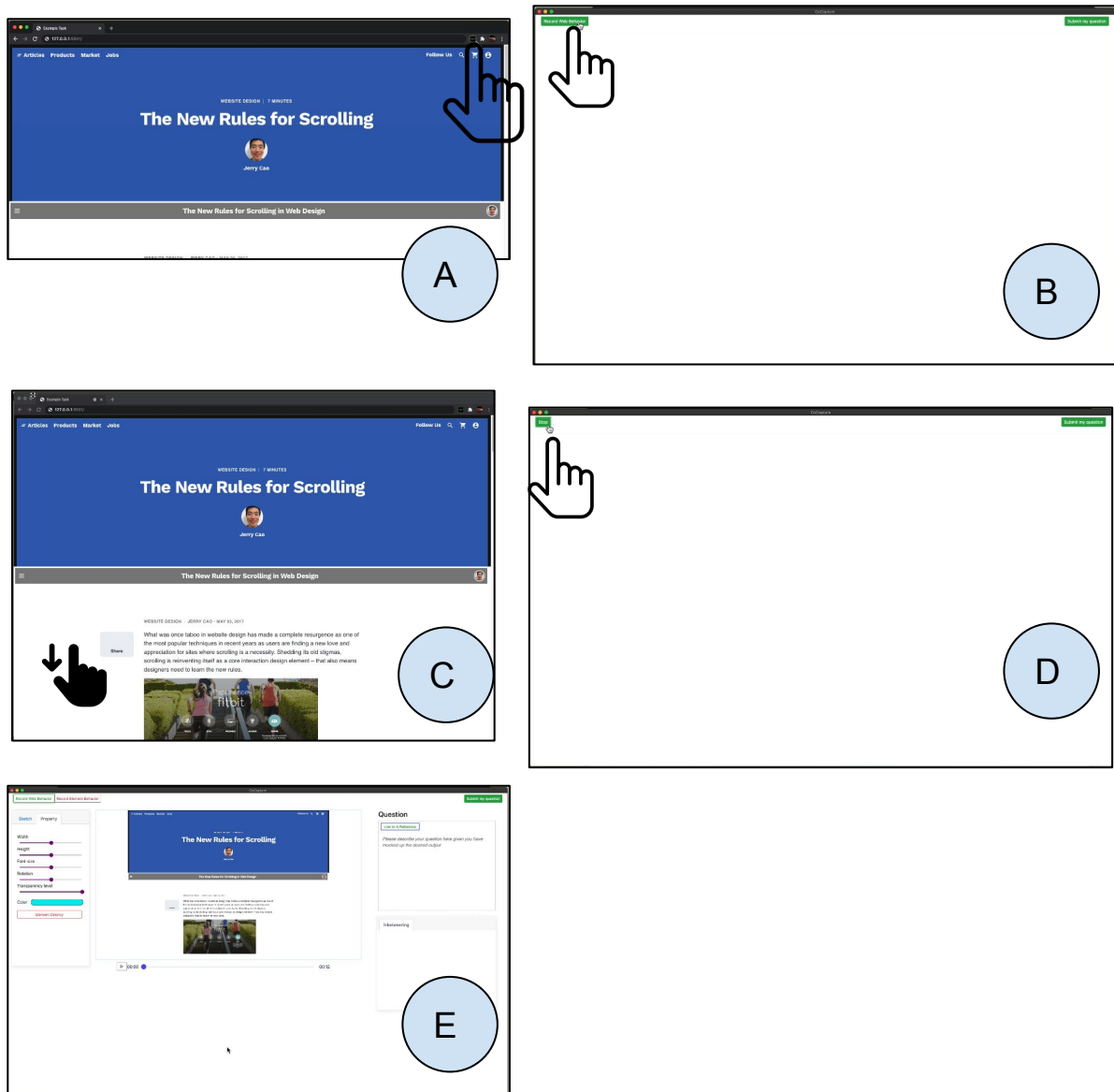


Figure 5.4: Five screenshots of Jerry's desired user interface at different scrolling percentages.

website, Jerry first clicks the Chrome extension in his browser to start CoCapture (Fig. 5.5.A) and clicks the “Record Web Behavior” button (Fig. 5.5.B) to record a demonstration of him scrolling through the original website as if he were the user (Fig. 5.5.C). Once finished, he clicks the “Stop” button, (Fig. 5.5.D) and CoCapture automatically reconstructs the demonstration replay and presents it on its main panel (Fig. 5.5.E). Jerry clicks the “Replay the mockup” button (Fig. 5.2.b2a) to watch the replay of his demonstration (Fig. 5.2.B), verifying that all the relevant context is captured.

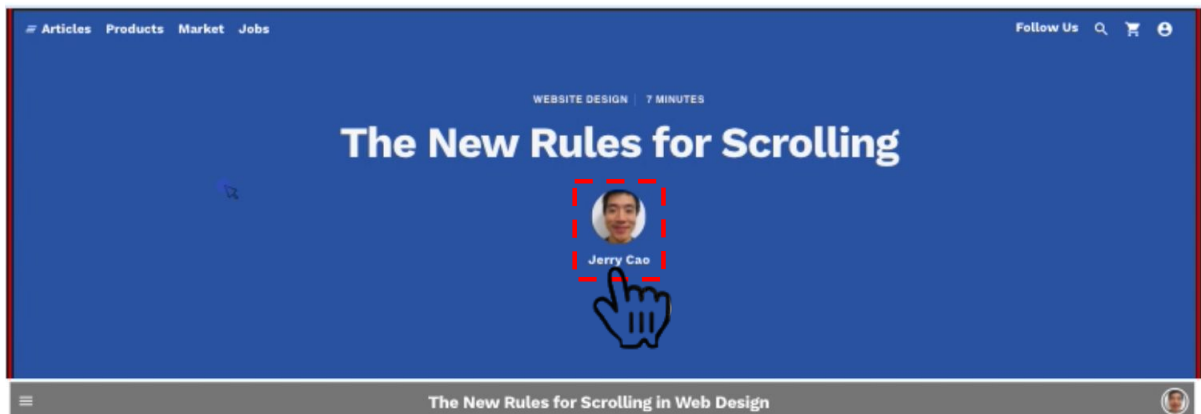
To create a mockup where his profile picture fades out down with the appropriate speed and distance as a user scrolls down the whole page (Fig. 5.4.B), Jerry first moves his cursor over the picture until he sees a dotted border highlight the correct element (Fig. 5.6). After selecting the element, Jerry presses the “Record Element Behavior” button (Fig. 5.2.A) and directly demonstrates the desired fading behavior. This behavior requires remixing two independent animations: the element moves down until the bottom half of the image is covered by the gray banner, and the transparency level of the element continuously decreases to half of its original



**Figure 5.5:** Five screenshots of the five actions that Jerry took to record his base scene.

value. Jerry creates these two animations by direct manipulation, seeing the changes immediately: he drags the element to the desired position (Fig. 5.2.b1) and adjusts the “Transparency level” slider value to half of its original value (Fig. 5.2.C).

The created animations are added to the “Animation List” (Fig. 5.2.D). Each animation has a set of operators, including an interactive range slider that represents options for the start/end time, replay, cloning, deletion, and preview. To set the recorded fading behavior to occur between the time when a user starts scrolling and the header moves out of view, Jerry scrubs the replay playbar to find these two moments from the base scene recording and then adjusts the animation sliders to align with them.



**Figure 5.6:** CoCapture highlights the element, the profile image, that the cursor hovers over.

After Jerry finishes creating the mockup, he adds a text description to his request, which includes references to the relevant DOM elements and animations (Fig. 5.2.E). Because Jerry knows he can use hypertext to link his description to the relevant artifacts, he writes a very short message and uses pronouns such as “*this* element” or “behave like *this*” (Fig. 5.7.1). Jerry selects part of his description and clicks “Link to a Reference” (Fig. 5.7.2) to select the DOM element or animation that he wants to reference (Fig. 5.7.3). To ensure the animation matches what he has written in the text, Jerry clicks the hypertext highlighted in blue and reviews all of the animation replays—both in the scene (Fig. 5.2.B) and in the “Inbetweening” panel (Fig. 5.2.F).

### 5.3.2 Design and Implementation

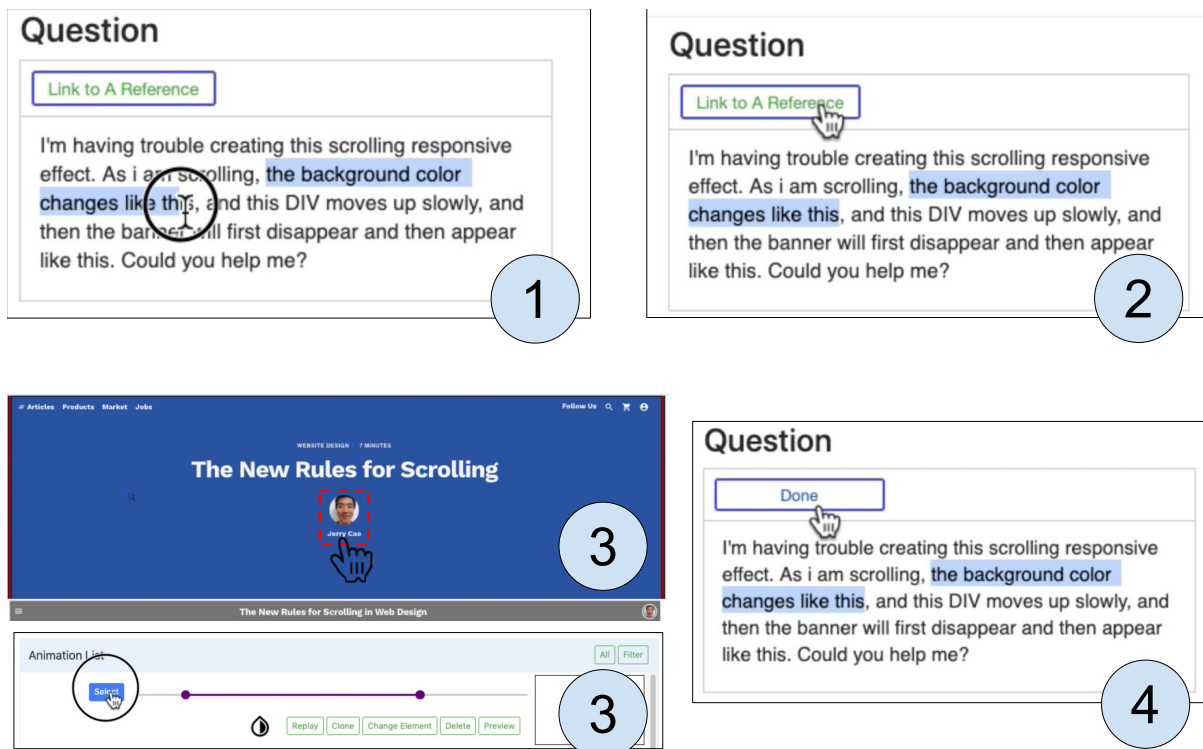
We describe the technical details of CoCapture in this section.

#### Step 1: Demonstrating Existing Behaviors

To quickly create visual context on top of an existing interface (DG1), CoCapture allows users to capture a behavior by demonstrating it on an existing website. This saves significant time and effort in creating a shared visual context—which we will call a *base scene*—as users can import arbitrarily complex behaviors from any website rather than needing to create animated visuals from scratch. When users demonstrate existing behaviors on a website, CoCapture captures all DOM changes (e.g., node creation, deletion) and events (e.g., mouse movement, browser window size changes). To capture this data, we rely on an open source library,<sup>7</sup> which in turn uses MutationObservers to track DOM changes [143] and stores the timeline of DOM changes.

<sup>7</sup><http://rrweb.io>





**Figure 5.7:** CoCapture allows users to link either an animation or a DOM element to part of their text to help make the description more useful.

The serialized DOM change sequence can be replayed on CoCapture’s main panel as if it was a screencast (Fig. 5.2.B). However, unlike a screencast (pixel-based), each frame in the replay still preserves the DOM tree structure from the original interface.

## Step 2: Animating the Desired Behaviors

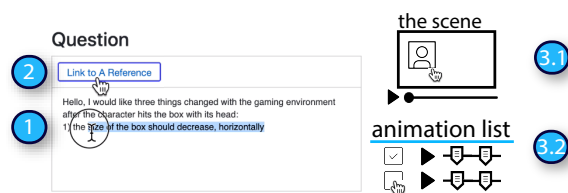
CoCapture includes a prototyping environment that allows users to demonstrate their desired UI behaviors as animations (DG2). Users can modify the replay of the existing behavior by directly manipulating the UI elements in the base scene, recording their changes, and augmenting the base scene with these demonstrations. The reconstructed DOM recording (i.e., the base scene) also preserves the UI states at each time point of the demonstration (e.g., DOM structure). It accomplishes this with the following steps and techniques.

*Manipulating and adding UI element(s) in the base scene.* CoCapture transforms all the UI elements from the original website into selectable elements that a user can directly manipulate. Users can select one or more DOM elements (by holding the Shift key) in the scene. As users hover over each element, CoCapture highlights the element with a red dashed border to ease the selection process (similar to the element selection feature in the Chrome Developer tool).





**Figure 5.8:** For each created animation, CoCapture provides information that prompts users about the actions they can take, including creating visual tags and replaying or previewing the animation. It also supports a set of operations to make the creation process easier and more accurate, including a range slider for time adjustment, as well as options for cloning, changing, or deleting an element.



**Figure 5.9:** There are three steps to create hypertext: (1) select portions of the text in the question description, (2) click “Link to A Reference,” and either (3.1) select an element in the scene or (3.2) check an animation. After this, the hypertext will be highlighted.

CoCapture also allows users to create low-fidelity sketches (which it stores as SVG drawings), import sketches into the scene, and manipulate them like any other DOM element (Fig. 5.2.C).

*Recording a desired behavior as a behavior mockup.* CoCapture uses the MutationObserver API to record the attribute changes (e.g., style changes) of the selected elements. Once they have started recording, users can edit selected elements’ CSS properties by adjusting the sliders in the relevant side panel tab (Fig. 5.2.C). They can also perform drag-and-drop operations on any DOM elements to demonstrate their new motion and position. All the manipulations will be represented immediately in the base scene. CoCapture automatically records and stores a continuous series of time-stamped snapshots, which will later be replayed as an animation.

CoCapture supports the modification of 10 commonly used element attribute types, including height, width, font-size, rotation, transparency level, color, hide (delete), visibility, x, and y. Future work can build on this list by connecting with Chrome Developer Tools or by extracting the existing properties of each element [188].

### Step 3: Remixing Added Animations

To help users accurately express and see the desired and existing behavioral information (N, DG2), CoCapture provides a set of features that users can use to remix, review, and fine-tune animations. The design of these features is inspired by the concept of “remixing,” an idea widely used in animation creation [121] and music editing [119]. First, each animation is listed below the base scene recording, with the ranges aligning with the exact moments the animation was started or stopped with respect to the base scene recording timeline (Fig. 5.2.D). Second, the set

of operations for each animation (the green buttons in Fig. 5.8) is designed to ease animation creation and adjustment. Users can replay the base scene (Fig. 5.2.b2.b), the remixed behaviors (Fig. 5.2.b2.a), and each animation solo (Fig. 5.8 “Replay” button) at any time. We describe the details of each feature below (all within the “Animation List” section in Fig. 5.2.D).

*Time and duration adjustment play bar.* The interactive range slider for each animation represents the starting and ending time of the animation relative to the original recording. A user can move the two handles of the range slider to modify the starting time and ending time. Because the scene’s play bar and the animations’ range sliders are visually stacked and follow the same time scale, users can easily remix an animation to align with others in the list. As the range changes, the duration of the animation also changes linearly.

*Replaying and deleting individual animations.* CoCapture allows users to replay and delete each animation by clicking the appropriate button below the range slider. When replaying, the elements that were selected when demonstrated will be highlighted with a solid red border (Fig. 5.2.b1). CoCapture replays an animation in two steps: it first resets the scene to the state at the starting time of the animation, and then it replays the demonstrated changes to the elements. Resetting the scene is necessary because certain behaviors are state-dependent (i.e., a banner only appears when the page scrolls to the bottom). Being able to replay individual animations from the correct state helps users precisely envision and reason about further steps. In addition, demonstrated animations can be deleted individually from the scene. This is helpful if the designer notices a mistake in the demonstrated behavior and simply wants to redo it.







*Animations at a glance with visual tag and preview.* To remind users of the animation type (i.e., edited element properties), CoCapture uses icon-like visual tags and a live preview for each animation. We use six different tags to represent the 10 different attribute changes. From top to bottom, Fig. 5.2.D illustrates an element’s change in color  (transparency level, color) and position  (x, y, rotation), and shows that it can be removed from the scene . CoCapture also includes resize (height, width) , font-size (font size) , and visibility  icons. The preview feature (The “Preview” button and preview demo in Fig.5.8) is also designed to help make animations more glanceable and easier to understand by showing a simplified version of the actual animation (N, DG3). Upon clicking the “Preview” button, the preview demo will play a simplified version of a looping animation in which each hollow square represents one relevant DOM element. This simplified animation preview design is inspired by Tufte’s minimalism theory for effective information visualization [192]. For example, the first preview demo in

Fig. 5.2.D is currently playing the transparency level changes of the background DIV.

*Cloning and changing elements.* CoCapture stores the animated elements and their mutation arrays independently. This allows it to apply the same set of mutations to different elements. Users can easily create one animation and use the “Clone” or “Change Element” buttons to repurpose it so that the behavior of one element can be copied to others.

*Filtering the animation.* To help explore the mockup details, users can click to select an element (e.g., profile picture) in the base scene and click the “Filter” button (Fig. 5.2.D) to view only those animations that were created for this element.

#### **Step 4: Communication With Visual References (Hypertext)**

To effectively communicate about the interface behaviors (DG2, DG3), CoCapture provides a lightweight text box (Fig. 5.2.E, suggested in the need-finding results) where users can easily and accurately describe and review UI behaviors with hypertext, a feature that links text description to the visual information. Similar to the hyperlink feature in common text editors, a user may select portions of the text, click the “Link to a Reference” button, and select any DOM elements in the base scene or created animations to link to the text (Fig. 5.9). Upon clicking part of the text with hypertext, CoCapture highlights and replays the referenced visual context with a solid-colored border (Fig. 5.2.b1). Meanwhile, to make an animation more glanceable, we designed the “Inbetweening” panel (Fig. 5.2.F) inspired by the technique of *inbetweening*—a computer graphics term describing the process of generating all the frames of a motion sequence given its first and last frames—to accurately communicate about animations [28]. Unlike the preview feature introduced before, the “Inbetweening” panel presents four key frames of the referenced animation using simple linear interpolation in between. More advanced interpolation calculation algorithms can be applied in future work to make it more expressive [167]. Similar to the preview feature design, each key frame represents a simplified state of the relevant elements (e.g., color, proportional size) using a hollow square. For example in Fig.5.2.F, the “Inbetweening” panel presents the four key frames of the profile image motion animation when the user clicks the hypertext attached to “slowly.”

## 5.4 System Evaluation

We conducted two initial user studies to evaluate CoCapture’s effectiveness to help users create, describe, and understand UI mockup questions. Primarily, we wanted to answer the following questions:

- Q1: Can designers ask a more accurate UI behavior question using CoCapture than with a text-based communication tool (e.g., email)? (Study 1)
- Q2: Can they also create the questions more quickly? (Study 1)
- Q3: Can helpers easily understand the questions in CoCapture? (Study 2)

### 5.4.1 Study 1 - Creating UI Behavior Questions

To evaluate question creation, we designed a two-condition, within-subject study. We recruited 15 participants (9 male, 6 female, age 25–30) with an average of 3.5 years of UI prototyping experience from a local participant pool. All participants had native or bilingual proficiency in English. Instead of using open-ended tasks, each participant was presented with four websites and asked to create one question per website regarding a predefined UI behavior issue. This helped us compare the description accuracy between conditions. Participants were randomly assigned to use either CoCapture or the tools in the control condition. In the control condition, we asked participants to use Gmail<sup>8</sup> and Google Drawings<sup>9</sup> to compose their questions. We chose these tools because of their low learning curve and similar functionality to the tools mentioned in the need-finding study. In the treatment condition, participants first watched a tutorial on CoCapture and replicated the example in the tutorial.

To recreate a situation in which the participants would naturally describe a new UI behavior on top of their websites, similar to a prior study setup [139], we asked participants to imagine themselves as the designer of the task UI, and also told them that a helper with domain knowledge (but no prior information about the task) would review their questions. We conducted follow-up interviews after each session. We compensated each participant \$20 USD for their time.

We asked participants to ask two questions per condition: one for Web UI behavior, and one for SVG game UI behavior. We chose these two categories because of their popularity and their difficulty. These questions’ complexity is often the result of highly dynamic transformation upon user input and the interplay among different DOM elements. Additionally, we wanted to ensure

---

<sup>8</sup>[www.google.com/gmail/](http://www.google.com/gmail/)

<sup>9</sup>[docs.google.com/drawings/](http://docs.google.com/drawings/)

our tasks and UIs were realistic, so we modified two commercial websites (Reddit<sup>10</sup> and Stack Overflow<sup>11</sup>) and also created two SVG game UI behaviors using an open-source project.<sup>12</sup> We designed these tasks using a rubric based on the common issues we found from our need-finding study. Similar to our need-finding study setup, we wanted to simulate a scenario in which participants have the desired UI changes in mind but no access to a visual representation of them. To avoid biases, for each task we gave participants both the existing UI and the desired UI (i.e., ground truth), and pointed out the differences between the existing and desired UIs, with visual annotation (e.g., cursor pointing) and determiners (e.g., “this part”), without offering specific descriptions. Participants could always access this information throughout the session but were not allowed to use any materials from the desired UI artifact (e.g., no screenshot of the desired UI).

## 5.4.2 Results and Analysis

We compared the accuracy of each question—meaning the number of satisfied requirements in the description—and the time spent asking each question between conditions. To measure the accuracy, we recruited an expert, a teaching staff member from a UI development course taught at the authors’ university. We adopted the evaluation method from prior work that measures the correctness of touch behavior implementation [151]. We provided the expert the task rubric, the existing UI, and the desired UI artifacts. The rubric included items corresponding to the UI behavior requirements we gave to participants, and all items contributed equally to the accuracy percentages reported below. The expert evaluated the question for each item by examining whether the item description was matched. We calculated (the number of matched items / total number of items) to calculate the accuracy per question. We used a two-tailed Welch’s t-test for our statistics analysis.

### Participants created more accurate questions with CoCapture

Participants were able to create more accurate questions in the CoCapture condition ( $p < .0001$ ) than in the control condition, resulting in average recall of 92.08% ( $\sigma = 12.17\%$ , CoCapture) and 54.05% ( $\sigma = 18.46\%$ , control) of the rubric items (Table 5.1) (Q1). The questions in the control condition included more words ( $\mu = 115.71$ ,  $\sigma = 68.83$ ,  $p < .001$ ) than the CoCapture

---

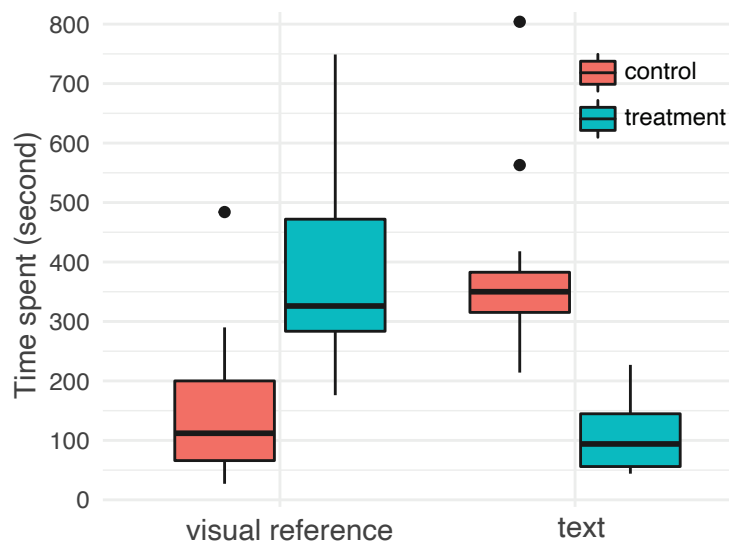
<sup>10</sup>[www.reddit.com](http://www.reddit.com)

<sup>11</sup>[www.stackoverflow.com](http://www.stackoverflow.com)

<sup>12</sup>[github.com/starzonmyarmz/js13k-2018](https://github.com/starzonmyarmz/js13k-2018)

	Control	CoCapture
description accuracy (%)***	54.05 (18.46)	92.08 (12.17)
# of words in description**	115.71 (68.83)	38.23 (18.16)
# of visual references ( <i>ns</i> )	2.29 (1.01)	2.53 (0.52)
# of application switches***	15.10 (6.98)	5.03 (1.46)

**Table 5.1:** Measurements from 15 participants using CoCapture and control tools (email + Google Drawings). Description accuracy (%) is calculated using (the number of satisfied items in a rubric/ total number of items). Visual references include images, sketches, and animations. \*\* indicates  $p < .001$ , \*\*\* indicates  $p < .0001$ . Their corresponding standard deviations are in parenthesis.



**Figure 5.10:** Time spent (s) on visual reference creation and text writing for the two conditions. The participants spent more time creating animations and less time writing the textual description in the treatment condition. For both cases, the differences were significant ( $p < 0.0001$ ).

question did ( $\mu = 38.23, \sigma = 18.16$ ), but no significant difference in the number of visual references was found ( $p > .05$ ). These visual references included images and sketches in the control condition and animations in the treatment condition. This indicates that CoCapture helps participants spend less effort describing the dynamic UI behaviors through writing. We found no sufficient evidence that using CoCapture would force participants to add extra visual information, which can be overwhelming for those who review the questions.

### Participants who used CoCapture spent less time writing.

One potential downside of using CoCapture is the time that it takes to create desired behaviors by capturing the base scene and adding the desired behaviors. Overall we have insufficient evidence

that using CoCapture will require more or less time to create a question (Q2) (Control (seconds):  $\mu = 645.68, \sigma = 377.39$ , Treatment:  $\mu = 482.96, \sigma = 190.77, p > .1$ ). We further observed and annotated the video to break down the overall time, allowing us to understand how participants used CoCapture, especially the time that they spent writing the description and creating visual references. We found that, in terms of time, the trade-off between the two conditions existed in authoring animations and writing textual descriptions. While the participants in the control group spent most of their time writing textual descriptions, the opposite was true in the treatment condition. Participants using CoCapture spent most of their time creating visual references via animated behavior (See Fig. 5.10). In the control condition, the participants' sketch activity included taking screenshots and sketching the desired output, but the limitations on what they could express led them to spend more time describing the behavior via text. These results suggest that CoCapture encourages designers to actually prototype the behavior visually, which increased the description accuracy, as opposed to describing it in Natural Language (NL).

An additional benefit of using CoCapture was that it required less context switching (e.g., switching between applications), as participants could write down their questions and create visual references in a single application. The participants in the control group switched more frequently between different applications, ( $p < 0.0001$ ) as shown in Table 5.1. This indicates that CoCapture can reduce users' cognitive effort by requiring less context switching across different applications.

### 5.4.3 System Usability and Study Insights

To better understand other usability benefits of or issues with CoCapture, we ran a thematic analysis on the interview transcripts with our own observations of participants' behavior patterns from the video.

#### **CoCapture is needed, useful, and easy to use.**

All the participants (15/15) recognized part or all of the scope, purpose, and value of CoCapture. For example, P15 summarized that:

*“To redo this [study task] in like [Adobe] After Effects, which is probably what you would have to use, you basically have to rebuild every single part of this. At that point, you might as well just wing it and see if you can pull it off. Because it's gonna take such a long time to develop the animation, which will be ridiculous. So yes, I think CoCapture totally makes sense.”*

Additionally, all the participants (15/15) gave positive feedback on CoCapture’s user interface, feeling it saved them effort on *“thinking about the description [or] writing the code”* (P10) and was easy to use, as *“it’s very similar to video or audio editing”* (P1). Furthermore, all participants used the hypertext feature. They found it *“super important”* (P3, P9), thought it helped *“save effort/time”* (P2, P5–P10, P12–P15) in describing behaviors that were difficult to explain via text, and said that it helped them to be *“more concise”* (P4, P11) when creating their questions.

### **Describing behavior without knowing the exact terms was a complex task.**

In the control condition, we found that some participants were not able to clearly describe certain objects in their email. Five participants struggled to find suitable terms for different interface elements (e.g., the game character). As a result, they either took screenshots of the elements and annotated them or described the object using its properties (e.g., black box). Four participants in the control condition said that they spent a great deal of time writing the question as clearly as possible to ensure that the helper would understand their requests. One participant even searched for the name of a type of game action to make the question *“clear enough for others”* (P4). This suggests that communicating visual information using text can be both time-consuming and difficult. In the treatment condition, we instead observed that participants were able to simply use demonstrative pronouns (e.g., “this”) with hypertext.

### **CoCapture helped decrease users’ mental processing costs.**

Participants in the control condition had to frequently switch between creating visual references and writing text (Table 5.1). In contrast, 11 participants in the treatment condition created the entire animation first and then wrote the question in CoCapture. CoCapture helped participants *“expend less effort thinking about phrasing and terminology”* (P7). One participant noted that *“once I had the mockup made, I can kind of just refer back to that. So it was easier”* (P5). This suggests CoCapture shaped participants’ workflow when creating questions, helping them organize the information and potentially better match the mental model through visual communication.



## **Capturing dynamic behaviors with static information was difficult.**

In the control condition, two-thirds of the participants (10/15) used Google Drawings to sketch on top of screenshots or create mockups of the desired interface from scratch. Three participants failed to adequately capture the scene in which the game character stayed in the air, as it required designers to operate multiple user inputs (i.e., capturing, jumping) at the same time. In contrast, CoCapture enabled participants to pause existing behaviors in the recording, making it easier to add consecutive or overlapping behaviors. The contrast between both groups' feedback paints a clear picture of the benefits brought by CoCapture's ability to manipulate elements of existing behaviors.

## **Requests in the control condition overspecified or underspecified question details**

When describing visual behaviors, it can be challenging to find the right specificity level. With the questions in the control condition, we observed participants' tendency to either overspecify or underspecify their questions. The larger standard deviation of the number of words (68.83) may be attributed to these varying levels of specification (see Tab.5.1). Additionally, we observed that overly precise or unnecessary details made the resulting description tedious and potentially more difficult for others to read and comprehend, as shown in the study below. *"My question [in my email] was longer because I kind of describe all the behavior. With CoCapture I kind of just say, hey, I want this [referring to an object] to resize like this [referring to an animation]. Instead of saying like, something much more specific"* (P5). In turn, underspecifying made participants' requests unclear, which was confirmed from the follow-up study (Study 2). This reflects the inherent challenges that exist in current methods of visual communication, as they only support text and static visual information creation, making dynamic visual changes difficult to specify. Our results suggest that CoCapture can help specify dynamic information by enabling animation creation and remixing.

## **5.5 Study 2 - Understanding UI Behavior Questions**

To evaluate CoCapture's effectiveness for helping people understand UI behavior questions, we recruited six participants from the same participant pool as in Study 1 as well as from Upwork,<sup>13</sup>

---

<sup>13</sup>[www.upwork.com](http://www.upwork.com).

to review the questions created in Study 1. There was no overlap in participants between the two studies. All participants had at least two years of UI development experience. Each participant acted as a helper and was assigned to evaluate four questions created by one participant (only one to reduce learning effect) in Study 1. Participants used the rubric (same as in Study 1) to rate each question from 1 (not clear at all) to 5 (completely clear). Before reviewing the two CoCapture questions, participants watched a 5-minute tutorial on the use of CoCapture, and they were also asked to try it to review an example question. After the evaluations, we interviewed each participant. We conducted Study 2 remotely using TeamViewer.<sup>14</sup> Each session lasted 30 minutes, and we paid each participant \$15 USD.

### 5.5.1 CoCapture Showed Potential in Facilitating Effective Comprehension of UI Questions

While we did not conduct any statistical analysis due to the small number of participants, the averages of time spent and clarity assessment showed promise that CoCapture can be effective in helping users comprehend questions. For the CoCapture questions, participants spent 247.25s ( $\sigma = 152.03$ ) evaluating them and gave an average rating of 4.21 out of 5 ( $\sigma = 1.32$ ) for clarity. In contrast, participants rating questions from the control condition spent 311.83s ( $\sigma = 176.12$ ) and gave an average rating of 1.40 out of 5 ( $\sigma = 1.54$ ).

### 5.5.2 CoCapture Questions Were Clearly Presented

In the post-hoc survey and interview, all six participants (S1-S6) found that the CoCapture questions were more clearly presented (Q3). One participant noted that *“it [CoCapture] is like combining what the email does the best with what video does best”* (S1), while another said *“it’s just like watching a video, and I don’t have to guess what the text is asking”* (S5). In contrast, participants thought other people might fail to accurately interpret some UI behaviors written in the control condition due to their lack of expressiveness or implicit assumptions. *“I know it’s common sense that the object will disappear when the character hits it, but not everyone knows that, so I can’t understand it when reading the question”* (S5).

We also observed that all participants constantly scrolled up and down the questions in the control condition (Gmail). S2 talked about the reason: *“I was scrolling up and down because*

---

<sup>14</sup><https://www.teamviewer.com/>.

*I was like, where did I see this information? It's not said very clear. I had to read the email a bunch of times to start building up my understanding of what it was asking for.”* For the CoCapture interfaces, by contrast, participants found that the hypertext feature helped them navigate the visual references more easily, lowering their cognitive effort. The preview and motion trail feature for each mockup highlighted the associated changes, guiding participants’ attention to the relevant information. *“I can actually see the visual like trail of the box moving associated exactly with the NL that it’s written in”* (S6).

## **5.6 Discussion**

In summary, we found that CoCapture helped participants in Study 1 (“requesters” from now on) effectively communicate about UI behavior issues by easing the creation of desired behavior mockups on existing websites and by clarifying NL descriptions with hypertext. This in turn made those behavior issues clear and explicit, and it made the descriptions easier to navigate for participants in Study 2 (“helpers” from now on). Consistent with prior theory [106], this finding indicates that clear visual context helps people ground communication. However, we observed that the dynamic nature of UI behaviors makes it more challenging for requesters to create descriptions, even with basic visual information (e.g., screenshots, sketches) as seen in [144].

### **5.6.1 Element-based Animations as First-class Objects**

As we have shown, requesters spent much more time writing text in the control condition than in the treatment condition, and the length of their written content was much greater. This is because common editor tools, such as Gmail in this study, only support linear sets of static content (e.g., text and images), which is limiting when trying to convey dynamic and interactive behaviors. One requester shared some of his working experience: *“Half the time I work with people where English is their second language. Being able to do it like this [CoCapture] would save a lot of time because the amount of time spent in trying to explain something to make sure that everybody’s on the same page is very time consuming”* (PI5). Communicating dynamic visual information via text creates a burden for requesters, who must describe their needs by dismantling the question into text and static images, which can result in underspecified description; it also burdens helpers, who must connect this disparate information in an attempt

to imagine what visual behaviors the requester wants to create.

CoCapture removes these burdens by enabling requesters to communicate their visual context as a whole without breaking it down further. Most helpers found that the UI behavior descriptions in CoCapture were more explicit, contextualized, and clearly presented. This is because CoCapture helped designers form a mockup of their requested visual behaviors to be delivered alongside their textual description. It maintained the dynamic information as a continuous and holistic view. Similar to the suggestions of prior work that code should be treated as a first-class object rather than a block of plain text [206], we argue that for UI behavior mockups, element-based animations should be treated as first-class objects rather than a combination of different pieces of information such as screenshots or plain text descriptions. Additionally, these animations can guide designers to write more concise and understandable questions than they could using the tools in the control condition.

### **5.6.2 Hypertext Helps Organize and Ground Communication**

Visual context and NL descriptions should co-exist during communication. With the desired UI behaviors created in CoCapture remixed holistically with the pre-built UI behaviors, the hypertext feature made it easier for requesters to explain their needs compared to the linear presentation in the control condition. Similar to prior work [196], the hypertext also served as visual cues to guide helpers in the second study to more easily parse and navigate each question. We noticed that in the first study, this feature freed requesters from following the linear order of a question, as they must with text-only descriptions. For questions that included multiple animations, the hypertext feature enabled helpers to prioritize those that were more important or complex, and it also reduced the burden of effort by helping them plan their response. To understand the questions, helpers constantly interacted with the remixed recording, referenced artifacts, and hypertext feature in CoCapture; when reading the emails, the interaction was limited to scrolling up and down. Together, our findings show that CoCapture shifted both requesters and helpers' main attention from textual to visual information, that is, the dynamic UI behaviors. It also changed their main workflow from linear to cross-referenced between text and UI behaviors, which we have shown to be effective for communicating UI changes that involve the dynamic transformation of multiple interface elements on existing websites.

### 5.6.3 System Scope and Limitations

CoCapture is most helpful for communicating UI behaviors that involve the dynamic transformation of multiple interface elements on existing websites. It is less necessary for simple element property changes (e.g., change the background color to red when clicking a button), or UIs that contain few elements or behaviors, in which case it is easy to reconstruct the artifacts from scratch using tools like Figma or Adobe XD. Additionally, CoCapture currently only supports web UI behaviors, but its interactive design and the study findings could be applied to or used for reference in other UI environments (e.g., mobile apps).

### 5.6.4 Future Work

Participants in both studies provided suggestions to improve CoCapture's efficacy and usability. Two requesters felt that CoCapture was similar to a video or audio editing tool, though it lacked some common features of those tools, like the ability to automatically save all changes or undo an action with a hotkey. Three requesters wished they could link the same text to multiple mockups or elements. This makes sense because designers often apply the same effect to different elements (e.g., an HTML class), or multiple behaviors to the same element (e.g., the box is moving while shrinking). Additionally, multiple requesters wished CoCapture indicated user inputs (e.g., click) during the demonstration on the base scene replay. Along with this, they suggested adding an alignment feature, like snapping to a certain timestamp when dragging the slider, to help them synchronize animations and the base scene by time, attributes (e.g., x position), or ratios (i.e., change/time). One way to realize this is to modularize the transition behavior like eXpresso [109] does. Three helpers suggested that CoCapture should highlight each hypertext as its corresponding element or animation is replayed. We will accomplish this by enlarging the hypertext text when the recording plays at the starting time of the associated mockup. CoCapture also preserves the original web DOM tree structure, and the style (e.g., layout), for each snapshot. Similar to Telescope [82], future work can use this benefit to further link UI elements and behaviors to related code snippets so that helpers can understand the code context cohesively and provide assistance directly.

## 5.7 Chapter Conclusion

In this chapter, we proposed an effective approach for UI designers to communicate about changes or issues of interactive UI behaviors on existing UIs. We designed and implemented CoCapture, which instantiates this approach to enable users to 1) easily (through direct manipulation) create animated mockups on top of arbitrarily complex web interfaces by demonstrating and remixing, and 2) effectively (via hypertext) communicate about these mockups with easy referencing techniques. Compared to existing approaches, we showed that CoCapture can help designers to create more accurate and more organized questions with rich context. In sum, CoCapture opens up opportunities for more effective user interface iteration by providing a natural and easy way of communicating interface behaviors.

# Chapter 6

## Supporting Remote Collaboration for Embedded System Development<sup>1</sup>

### 6.1 Introduction

On-demand collaboration occurs in all sorts of system development tasks, such as embedded systems. Compared to software development, which was the main focus of the previous three chapters, embedded system development is unique in that its tasks require people to operate physical artifacts in a physical environment. The rise of the maker movement has led to a growing number of developers who prototype and program embedded systems (e.g., Arduino), which is a complex task that requires knowledge and skill sets in both programming and assembling hardware for circuits [18]. As I have shown in previous sections, developers often rely on support from others. Ideally, people will share a workspace during collaboration (i.e., in-person and synchronous), which helps ground communication in shared understanding [58, 66]. However, collaborating with other people over distance and time (i.e., remote and asynchronous) has become more common in recent years. In fact, my prior studies [40] have demonstrated that the on-demand asynchronous help model can help developers complete more tasks than the synchronous model could. While the HCI community has introduced many remote collaborative tools, such as my two systems (i.e., CodeOn, CoCapture), they often fall short when supporting *remote* and *asynchronous* collaboration for physical tasks, such as connecting circuits [95].

Specifically, there are two shortcomings of current communication tools. **First**, because collaborators are not co-located, remote helpers often lack access to inspect and change the

---

<sup>1</sup>Portions of this chapter were adapted from [36]

physical artifacts. To address these challenges, Heimdall allows instructors to remotely inspect, measure, and rewire students' circuits [95]. However, it requires students to build their circuits on a specialized workbench, limiting the size of students' projects and prohibiting them from working in parallel when seeking help. In addition, teachers cannot directly manipulate the circuits, which limits their support to only a few types of tasks. **Second**, providing up-to-date feedback is necessary but challenging when collaborating asynchronously, as the state of the circuits may constantly change. To address this challenge, early effort has been mostly spent on improving the efficacy of communication via audio and video conferencing techniques [58, 108]. More recently, new techniques such as AR and Virtual Reality (VR) have emerged, aiming to provide a more "present" and immersive collaboration experience to facilitate and extend the existing communication channels of the physical world [152]. However, these techniques only improve communication, and users are still unable to perform physical activities that arise naturally or are required from social interactions and physical task collaboration [25, 93]. Furthermore, all physical manipulation must still be completed solely by the requester. Additionally, due to the nature of asynchronicity, by the time the requesters receive a response, the circuit connections and setup of the embedded system might be updated, making it difficult to integrate the response. Unlike the process for software development, it is often infeasible to automatically track meaningful changes and merge responses on physical artifacts due to the unstructured spatial information.

Therefore, we asked: **How can we design an embedded system support tool that enables remote helpers to directly manipulate the embedded system circuits?**

Teleoperated robots might be one approach as the technology becomes less expensive, more powerful, and more reliable. These robots have become more prevalent in homes and businesses, allowing visitors and business partners to be present without the need to travel. Studies have shown the potential of teleoperated robots in extremely complex tasks such as surgery [12]. However, the use of these techniques in an on-demand collaboration setting for embedded system tasks has remained largely unexplored, including concerns about privacy, safety, and helpers' expertise with teleoperation when the robots are used in other daily tasks.



## 6.2 Study Design

To answer these questions, I conducted semi-structured interviews and an observational study with 11 participants who had experience with remote collaboration for embedded system development. We recruited our participants via purposive and snowball sampling. We reached out to local makerspaces, embedded system research labs, and online crowdsourcing markets (e.g., Upwork, Instructables) with an email including a short description of our research purpose and a background survey about their relevant embedded system development and collaboration experience and demographic information. Each study session was conducted remotely via video conferencing software and lasted 60 to 90 minutes. For their time, each participant received a thank you gift commensurate with the market rate for developers in the authors' city (\$50/hr) after the study.

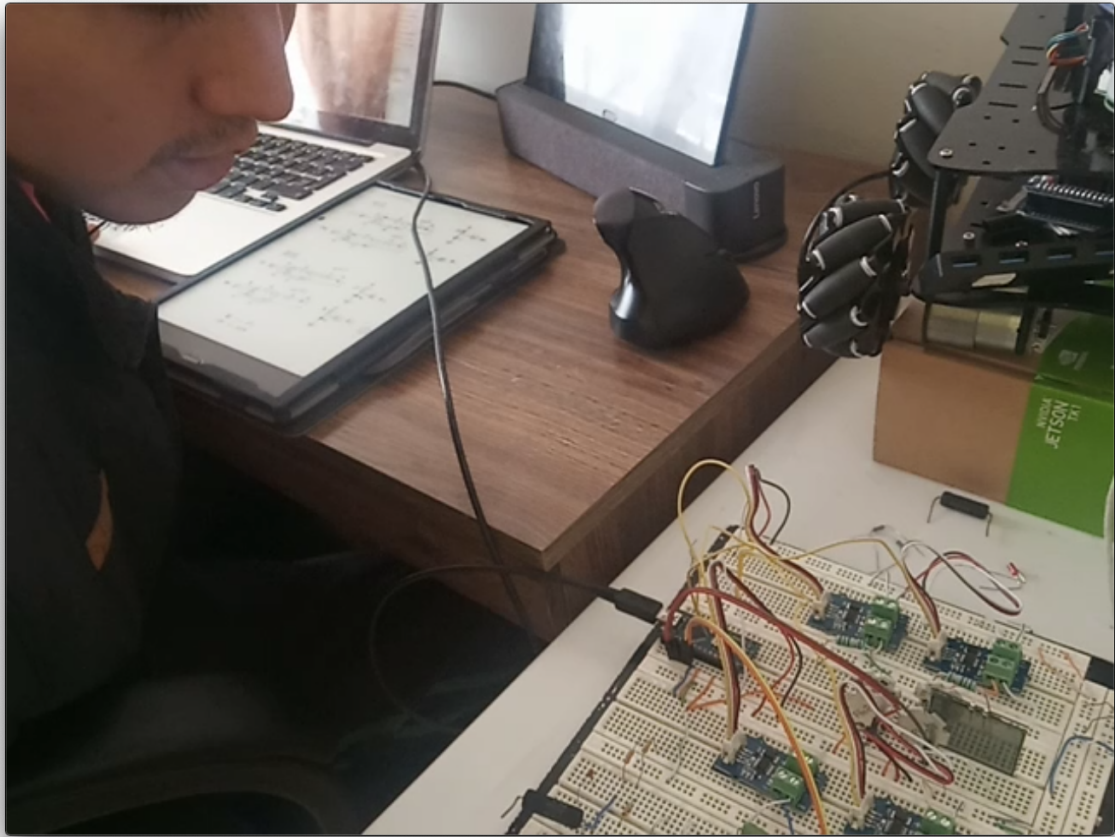
### 6.2.1 Semi-structured Interview Protocol

We designed the semi-structured interview based on our research questions mentioned in the introduction. We began by asking interviewees about their general practices for on-demand help seeking during embedded system development, such as the tools they used, the resources they relied on, the type of information they looked for, etc. We asked what they liked and disliked about these practices, and how they wished things could be improved.

### 6.2.2 Hypothetical Assistant Study Protocol

After the interview, we asked participants to work on their own embedded system projects on their workspace for at least 20 minutes, and to record a video of the session to be sent to us afterward (Fig. 6.1). We instructed participants to imagine that they had an intelligent assistant nearby who was capable of assisting any requests that they expressed verbally.

To ensure that participants were not constrained by their prior experiences interacting with an intelligent assistant, we remotely presented a proof-of-concept teleoperated robot arm prototype to help convey to our participants what the intelligent assistant might look like. The concept of using a robot arm as an intelligent agent was based on our lessons learned from literature review and informal conversations with multiple students, makers, and researchers who had experience with remote collaboration for embedded system development. From these lessons, we found that requesters often need to carry out physical tasks on their end during remote collaboration to

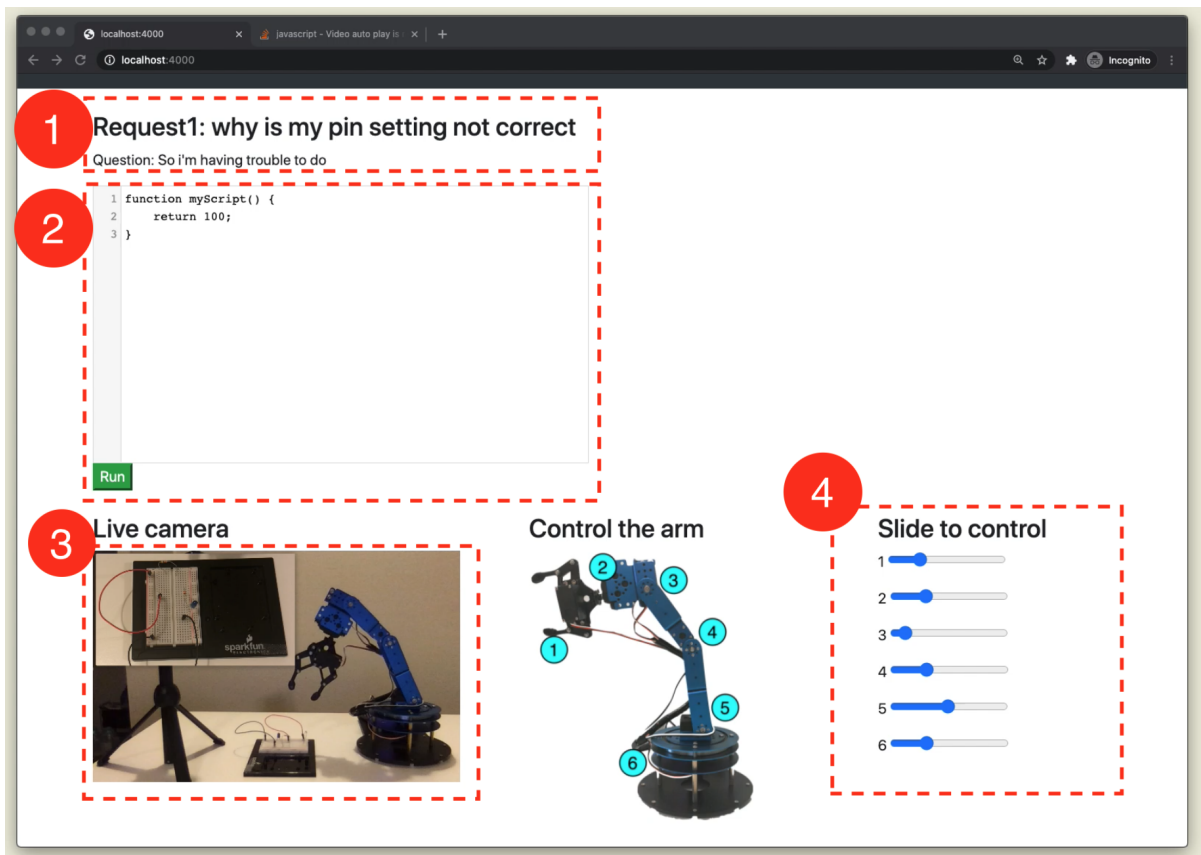


**Figure 6.1:** A screenshot of a participant making a request to the hypothetical assistant while working on the hardware part of the embedded system project during the study.

implement tests and changes according to helpers' reasoning, such as measuring the currents of a component or rewiring circuits. While these tasks are easy to perform, requesters spent most of their time realizing remote helpers' requests. Because of this issue, and because grounding communication for physical tasks via video conferencing tools has shown to be inefficient [58], we envisioned a robot arm that allows helpers to teleoperate and perform tasks on their own.

To help familiarize participants with the workings of a teleoperated robot arm, such as how a remote helper could control it, we designed and demonstrated a web-based, teleoperated DIY robot arm that aims to allow remote collaborators (helpers) to perform inspecting, wiring, and even coding tasks upon the end-user developer's (requester's) requests. The system consists of two parts: a robot arm (Fig. 6.3), and a web user interface (Fig. 6.2).

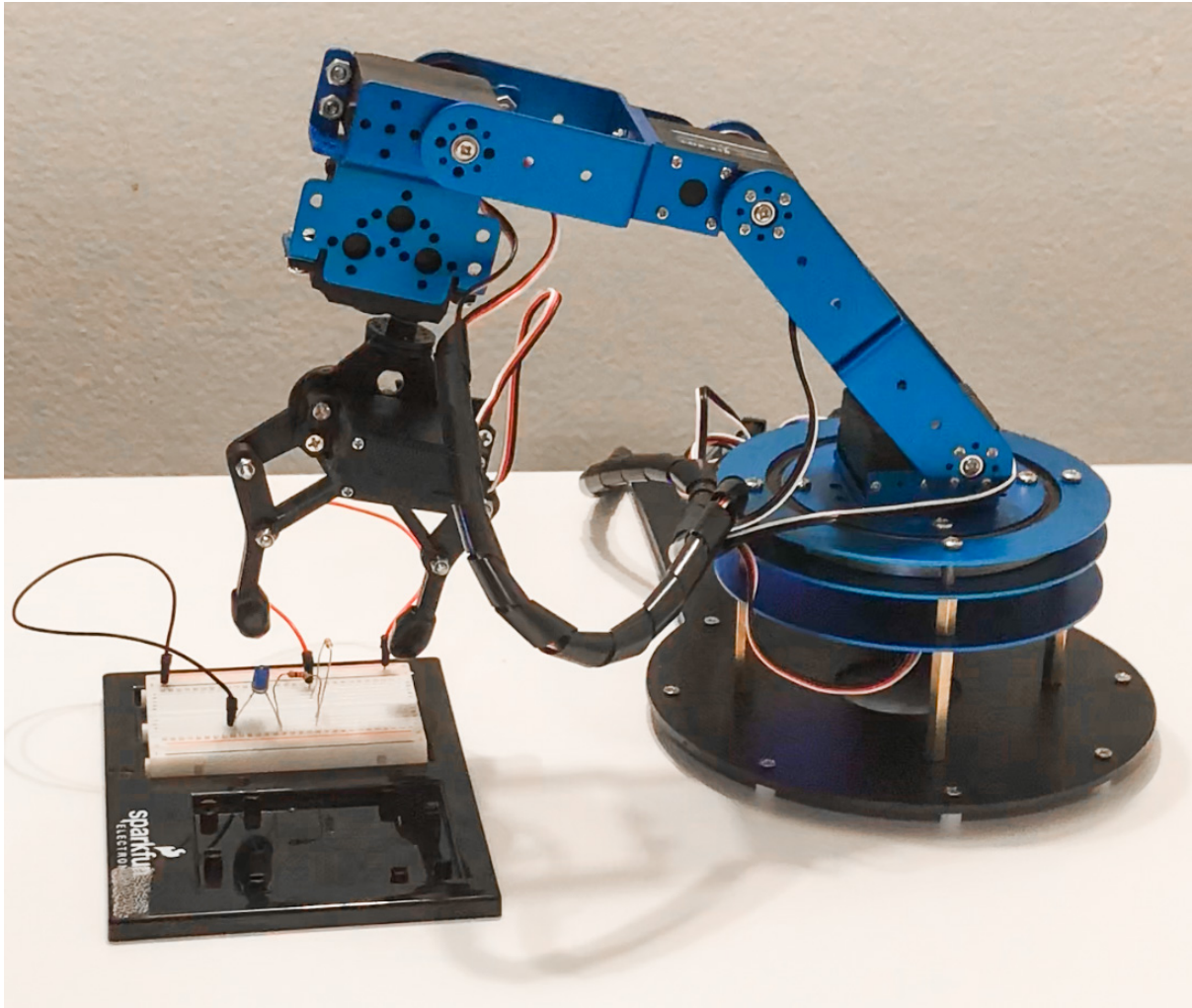
**Robot Arm** To enable physical manipulation, we used an off-the-shelf robot arm (USD \$59.00) [91] with 6 degrees of freedom that could perform simple pick-and-place tasks. The six servos are connected to an Arduino Uno board, which is controlled by a web UI.



**Figure 6.2:** Control web UI. It includes four parts: 1) a natural language task description, 2) a code editor that includes the task-relevant code, 3) two fixed-position live stream camera views, and 4) a 6-slider control panel that corresponds to the 6 servos of the robot arm.

**Web User Interface** To enable remote control, we designed a web UI that consists of four parts (Fig. 6.2): a natural language task description that specifies the end-user developer’s request; a code editor that includes the task-relevant code; two fixed-position live stream camera views, including one top-down view and one side view; and a 6-slider control panel that corresponds to the 6 servos of the robot arm. We used two Logitech C270 cameras that capture 1280x720 resolution in live-viewing mode. We adopted a fixed-position camera view rather than a moving and remote-controllable camera view, as prior studies have shown that users may have privacy concerns with moving and remote-controllable cameras such as drones [200, 197] and teleoperated robot arms [29]. We decided that a fixed-camera view would minimize users’ privacy concerns without sacrificing the usability of the arm.

After showing a demo video and explaining the system details to them, we also asked participants not to limit their requests based on the system’s capabilities (e.g., speed, size, precision) within its current version. This allowed us to better understand the types of questions that they would want to ask an intelligent assistant in the “best case” scenario if there were no



**Figure 6.3:** A photo of the robot arm performing a wiring task

limitations on time, compensation, or capability. We hoped this kind of investigation would give participants a more realistic impression of the technology while allowing us to probe their perceptions more naturally, particularly with participants who were not familiar with the teleoperated robot technology [197]. While working on projects, we asked participants to video record their workspace and themselves. This recording provided us spatial context information for our analysis (e.g., where a participant looked when making a request). After the hypothetical assistant study, we prepared a follow-up interview with the participants, asking basic information about the projects they were working on and their opinions on our approach.

We used an automated transcription service to transcribe all the interviews. We went through the transcripts to correct machine translation mistakes and exported the transcripts to a Google Doc. Through iterative coding, we then divided the interview responses into major themes using an inductive analysis approach [51]. We lightly edited interviewees' quotes for readability.

PID	Gender	Age	Occupation	Programming skill	Electronics skill	Collaboration experience
P1	M	31 to 36	P	7	6	6
P2	M	24 to 30	P	6	6	5
P3	F	24 to 30	P	6	6	7
P4	F	24 to 30	P	5	6	5
P5	M	24 to 30	P	5	6	6
P6	M	24 to 30	P	5	5	4
P7	M	24 to 30	P	6	7	4
P8	F	24 to 30	P	5	5	6
P9	M	24 to 30	S	6	5	4
P10	M	24 to 30	S	6	5	7
P11	M	18 to 23	P	6	7	5

**Table 6.1:** Participants' background information. For the Participant ID (PID) column, P1-P11 represent the 11 embedded system developers. For the Gender column, M = male and F = female. For the occupation column, P = professional developers (paid to develop embedded system projects) and S = student developers. Programming skill, electronics skill, and collaboration experience are on a 7-point Likert scale where 7 means expert and 1 means no experience.

### 6.2.3 Participants

Table 6.1 lists participant information across multiple dimensions, such as gender and age. The 11 participants had two occupations: professional developers who got paid to develop embedded system projects (9) and student developers who identified themselves as hobbyists, makers, or researchers of embedded system development (2). We will use P1-P11 to represent the 11 participants.

## 6.3 Findings

We have divided our findings into three parts: 1) current practices and challenges of on-demand collaboration in embedded system development, 2) request categories for the hypothetical assistant, and 3) suggestions for the hypothetical assistant.



### **6.3.1 Practices and Challenges of On-Demand Remote Collaboration in Embedded System Development**

#### **Relying on archived information is the most common method, but it is often non-personalized and outdated**

Unsurprisingly, all 11 participants reported that the most common method to seek for help is relying on existing documents (e.g., component datasheets), blog posts, or existing answers provided on Q&A forums. While there is a lot of this type of information out there, a crucial issue that participants also pointed out was that they often can only find information containing answers that are similar to what they need instead an exact match. This non-personalized help seeking might lead to frustration, delay, or extra cost.

#### **Seeking on-demand assistance from online crowds is also common, but the response time is slow**

When asked how they seek support other than using archived information (e.g., datasheets and/or existing answers on Q&A forums), almost all the participants (10/11) mentioned that they would directly ask questions in online forums such as Stack Overflow or relevant chatrooms (e.g., Discord, Gitter) to get assistance from an online crowd of experts. When asking questions, participants mentioned that they often include visual information to help communicate their requests (i.e., video demonstrations of their circuit boards, images with annotations, or circuit design diagrams).

Although there are many embedded system experts in these online spaces, developers often must wait a long time for someone with relevant knowledge to see the request and respond. This is consistent with prior work [132]. For this reason, participants wished that these platforms would group experts based on their backgrounds or specific knowledge.

*“If the forum could be separated into different areas for people that work in specific devices, and kind of build small communities or forums [of people] that are working in the same thing, [they] could exchange information more frequently. (P3)”*

#### **Divide projects into small parts to reduce the need for direct collaboration**

When asked how they seek support for hardware-related requests, participants stated that they would organize the task distribution to reduce the need for direct collaboration among

team members, as they found collaboration more difficult with physical components involved. Specifically, they would break projects into small parts such that each team member would work on their own part (e.g., one sensor). Once each part was ready, the team lead would integrate them into the final project.

*“To avoid having multiple people’s relying on the same hardware, we tried to split up the projects into hardware to-dos and software to-dos, especially if it’s a common framework. [For example], working with the Raspberry Pi and writing Python that’s on the Raspberry Pi...the Python script was a task for an individual. I’ve been mailing those out to different individuals, so that they can program and connect the hardware on the development kit, write up the code for that component, and then I can take all of those pieces in and put it all together for the whole system. (P5)”*

### **Known expert assistance is more personalized, but communication setup remains tedious**

When seeking support from an online crowd is inefficient, most participants (9/11) would rely on experts that they know personally, including professors (P9, P10), colleagues (P1, P3, P4, P7), and friends (P5, P8, P9-P11). They would conduct more direct and personalized help-seeking sessions with existing communication tools, such as Zoom, Skype, Slack, email, etc. They would also share the relevant document, such as code or PCB design, which they would not feel comfortable sharing with online strangers.

During real-time video chats, participants said they would often point their camera at the physical devices, such as their circuit boards or oscilloscope, while explaining what they need help with.

*“When we run into issues—like, big issues—or we need to see if a signal looks right on the oscilloscope. We’ll ask him [professor] to chime in. He doesn’t have the development kit and the sensor in front of him. And so we’ll often have like, a Zoom meeting. And we’ll like, talk about it if there is like, an environment that we can debug in, then we’ll show that and we’ll kind of step through the code. (P10)”*

With asynchronous communication, participants would include visual information such as videos or images of their circuits along with annotations and explanations. The helper would then respond to them with annotations on the image along with explanations.

*“Right now my advisor and I, we had a discussion over Slack over debugging. And basically*

*what ended up happening is like, I drew the diagram on my iPad and then I sent him a picture of the circuit and then I use colors to draw where I'm plugging in the scope. And then he sent me like, a marked up image back. But that's like right now, because we don't have any of these other tools that facilitate a better way of doing this. (P2)"*

However, unlike software development where sharing code and text is convenient with existing techniques, communication about embedded systems is more complex. Participants reported that their setup—with multiple physical components spread out across a workbench—felt cumbersome to explain to remote helpers. In synchronous communication, participants often want to point their camera at the physical components to highlight the issues they have.

*"What I have to do is, I have a Zoom on my laptop, and I rotate the laptop so that he can see what I'm doing on the screen while I'm modifying the hardware. (P7)"*

*"If I'm holding the circuit for minutes, it starts to bother my hand. I would have to like, hold it up and point to different things that I'm talking about. I just physically can't hold them long enough to communicate that. (P8)"*

In asynchronous communication, participants also found that the complexity of the circuits made it inconvenient to provide visual context.

*"I have another board that's more—from a circuit point of view—complicated. And you can try taking pictures of it. But when wires overlap, it's really hard to read a picture, because you're often like rotating a board to see different angles of it. (P11)"*

### **Better viewing and annotation tools would be helpful**

When asked what techniques would enhance their help-seeking experience, more than half of the participants (6/11) mentioned that they wished for better viewing tools so that “my collaborators can see where I am pointing at [and] looking at when speaking” (P4). A third of the participants (4/11) pointed out that they had to provide additional visual cues during remote collaboration as compared to in-person communication, including “gestures” (P1, P6) and “words” (P9, P10) to remedy the insufficient context caused by the limited viewing angle of their camera.

*"I'd want Google Glass. I don't care about the heads-up display part—like, that's important, but I would want something where people can see what I'm looking at. And they can see like, the way that the wires are connected to each other. Or like even a camera that's just pointing to my work surface. (P6)"*

With asynchronous communication, participants also wished to have better annotation tools



to exchange visual context. With current annotation tools, participants often need to record a video of their physical components while narrating their requests, or to draw a diagram of their wire connections with annotations. Meanwhile, helpers would often build on top of the video or diagram by annotating their suggestions, which can be tedious when the diagram is complicated.

*“I send them what I wrote—I drew the diagram on my iPad. And then I sent him a picture of the circuit, and then I use colors to note where I’m plugging in the scope. And then he sent me like, a marked-up image back. But that’s like right now. Because we don’t have any of these other tools that facilitate a better way of doing this. A lot of the annoying things that we’re doing is like, for instance, if we were physically together in person, I’d be able to draw the circuit. But if it’s a more complicated circuit, it’s a lot harder to like send detailed drawings of the circuit. So we’re going—we’re using professional software like Altium, a PCB layout software, to draw a circuit, because that’s the way that we’re facilitating remote collaboration right now. But in the real world, like nobody would do this. Like nobody would brainstorm using a professional tool. (P9)”*

### **Remote assistance would improve if helpers could directly manipulate circuits**

Interestingly, more than a third of participants (4/11) mentioned that they wished for a way the remote helper could simply operate on their devices to save participants the redundant effort of carrying out described tasks. This suggestion is beyond what prior work has explored, as research has proposed tools for better guidance rather than direct manipulation of physical devices.

*“Managing the hardware [is difficult], even if it’s something as simple as like, I need to press a button...Oh, which wire? Oh, the red one. Which red one? [There] are three. Oh, the one on the bottom? Like, it’s—it’s a lot more difficult. And when you’re in person, they can just move that wire, right? (P8)”*

Additionally, multiple participants (4/11) reported experience as helpers. Because of insufficient domain knowledge, some requesters might not be able to integrate or even understand the responses that they have received from their helpers or online resources. Partially, this is also due to the issue of experts’ “blind spot,” where people with greater expertise tend to make assumptions about student learning that turn out to be in conflict with students’ actual performance and developmental propensities [104]. In practice, the requesters might ask for follow-up questions even if the helpers have moved on to other tasks, or they would integrate

the responses incorrectly, causing further project delay.

*“For me it is easier to understand what is wrong with them [requesters]. But usually they cannot follow my work. Sometimes I spend a lot of time in supporting the clients [requesters]. Sometimes they [requesters] make the incorrect connection [on] the incorrect components usually. How you place the components is also important; maybe they [ruined the hardware]. So, when working with hardware, things are very sensitive. It is not just code; it is more important to place things correctly. (P4)”*

However, we also found evidence that this “blind spot” phenomena might not occur by accident. Two participants reported that they wish that helpers could “be there to put it [the circuit] in my hand” (P5).

### 6.3.2 Participants’ request categories for the hypothetical assistant

To make it clear when they were asking for support from the hypothetical assistant, we asked participants to say the word “agent” before they make a request. Because the agent is hypothetical, we asked participants to resolve their requests themselves after making them. Once the sessions were complete, we coded participants’ requests and found they fell into six major categories. Table 6.2 shows the description of the six categories, the number of sessions in which each request type appeared, and the requests’ overall frequency. We explain each category in more detail in this section. We will reference these categories in the design section.

Description	# of sessions (out of 11)	# / session
Measurement aids: Participants measured the physical component relevant signal/data	11	3.3
Physical aids: Participants asked for physical activity support	11	5.8
Memory aids: Participants sought specific information related to a physical component or code	8	4.2
Explanatory requests: Participants sought examples or explanations of their circuits or code	7	3.1
Debugging & bug fixing: Participants sought solutions to system errors	9	3.9
High-level strategic guidance: Participants sought high-level guidance to approach problems	6	2.8
Wire refactoring: Participants asked for circuit connection improvements	3	1.7

**Table 6.2:** Common query types observed during our hypothetical assistant study, with corresponding frequencies. Each of these query types suggests a support role that remote embedded system development assistants can play in future systems. “# Sessions” indicates the number of different sessions that each type of question occurred in, while “# / Session” indicates the number of queries that referred to solving one of these query types, on average.

## Measurement Aids

All the participants asked the agent to measure and display data or signals, such as the current or voltage of their circuits. For example,

P4: *“Check the GSM modem voltage pin.”*

P1: *“Can you check the borders in GPS and check their serial data?”*

P1: *“Is this device sending data to the server? Or is there any problem with this data-sending issue?”*

## Physical Aids

All the participants also made *physical aid* requests to the agent, asking it to either perform physical tasks for them, such as grabbing components, or to provide assistance on tasks they could not perform alone.

P7: *“Can you hold the circuit board tight so I can solder something on it?”*

P10: *“Please connect my Arduino to my computer.”*

## Memory Aids

As with physical aid requests, more than two thirds (8/11) of the participants made requests to look up the datasheet or relevant documentation of their devices, as well as APIs for programming libraries. For example,

P11: *“For this device, how much force is concealed during operation?”*

P5: *“Pull up the datasheet for the FPGA; I’m trying to figure out...how much current this microphone is supposed to take, because it’s taking up more current than I was originally expecting.”*

## Explanatory Requests

Close to half of the participants (5/11) sought explanatory support from the helper. They asked the agent to help them understand and learn things relevant to their projects.

P1: *“What does ‘Nordic DFU trigger interface was not found’ mean?”*

P6: *“What is the difference between VDD and ground?”*

## Debugging & Bug Fixing

Along with explanatory requests, most of the participants (10/11) made debugging requests. They either wanted to know where the bug was.

P9: *“Why am I getting this error?”*

P2: *“Can you find the short?”*

P11: *“I did not hit the reset button, but I still see the device in the drop-down menu, and the memory layout is blank. Why is that the case?”*

## High-level Strategic Guidance

Half of the participants (6/11) also got stuck at some point during their development, in which case they requested high-level hints or guidance from the agent. For instance,

P7: *“How do I delete a characteristic that I added to generic access instead of the battery service?”*

P11: *“Agent, I am trying to hit the delete button, and I am not getting the battery level to lead it. What else should I do?”*

## Wire Refactoring

Three participants also made “wire refactoring” requests. As with code refactoring, these participants wanted a cleaner and easier way to manage circuits so that they would not have to worry about not being able to find or measure certain components on their circuits.

P9: *“Can you clean up this voltage divider?”*

P5: *“Could you clean like, my circuits? They are like, super messy right now.”*

### 6.3.3 Findings and Follow Up Interviews

Overall, each participant made 25 requests on average (max: 54, min: 8). Participants often phrased requests in a way that required knowledge of their hardware (e.g., circuits), the specific context (e.g., where they were pointing), and their code base. We found that only 12% of participants’ requests were “self-contained” (e.g., What is the difference between VDD and ground?).

After the hypothetical study, we asked participants to provide a high-level summary of their projects and their approximate progress out of 100%, as described in Table 6.3. Knowing the

PID	Project Description	Progress (100%)
P1	A vehicle tracking system	80-85%
P2	A system to transfer data between devices	10%
P3	A privacy protective device	80%
P4	An agriculture monitoring system	60%
P5	A signal positioning system	70-80%
P6	A power level control system	90%
P7	A system to control multiple power supplies	40%
P8	(private)	(private)
P9	A system to control home smart devices	15%
P10	A software program to debug a micro-controller	20%
P11	A system to monitor sensor input data	10%

**Table 6.3:** Participants’ project descriptions and progress for their hypothetical assistant study. P8’s project contains private information.

project relevant information might help us know what type of project and at what stage of the project certain requests might occur (e.g., projects that are close to finishing might need more physical and measurement aids for testing). We left one project description out (P8) due to its privacy policy. As we requested, all of their projects involved both software and hardware. We also emphasized that the agent could handle both software and hardware requests.

To further explore the idea of using a robot arm agent as an assistant, we gathered participants’ opinions on the advantages and disadvantages of a hypothetical embedded system assistant, as well as any concerns. Every participant liked the idea of having a remote helper to control a robot arm. They felt the approach could help a great deal, especially given the mobility of the arm and the camera, which could basically serve as a remote surgeon. However, participants reported three main concerns that could help guide the future design of the system.

### Privacy and safety issues

The most cited concern among participants was privacy issues (10/11). They worried about their project ideas being stolen (P1, P4), their conversations with others in the room being overheard (P1, P6, P11), their face or room being seen (P4, P11), or the robot arm moving freely around in the room (P6, P9).

*“If there was like a battery source in the robot to keep it going after I unplug it that would be concerning, because it could essentially navigate or move to see any part of my room. (P6)”*

*“I wouldn’t feel comfortable if it was [a] random, like, Mechanical Turker doing it. Or I*

*would still feel even uncomfortable if it was even another undergraduate. Like, it just seems very impersonal. (P5)”*

To mitigate these concerns, participants suggested integrating visual privacy-protecting techniques into the system: “only have the device that I am pointing to visible, and blur the outside view” (P4). Prior work has studied users’ perception regarding different blurring methods applied to a movable robot arm and suggested that using an image depth filter (i.e., objects extremely close to the camera appear black, whereas farther away objects are lighter) is perceived to best protect privacy [101].

Three participants also worried that the robot arm could damage their devices (P8, P9) or introduce interference (P6).

*“You’re using a microphone sensor, and you’re trying to check for [a] certain frequency of something. And you would like the robot arm to place a probe at a point...to check the frequency. I wonder if the robot is going to cause interference [in] the project, like sound frequency or like mechanical effects on the performance. (P6)”*

Prior work also studied the performance of various control settings when humans control a robot arm. They suggested that for users with no background in robot control, allowing the robot to have some autonomy over the arm can help reduce error rates [122].

### **Uncertainty about precision, flexibility, controllability of the robot arm and helpers’ expertise**

More than half of the participants (6/11) were also concerned about the precision and flexibility of the robot arm, saying that they would have to first know how capable the arm was before using it on critical things: “I don’t necessarily need help in placing circuits, more like asking ‘why’ or questions about parts of the circuit. But if the robot had the dexterity, like if there was no physical limitation behind the robot being able to do this, I would be very positive towards that” (P9). Two participants had some experience with robot arms and were concerned about the controllability of the robot arm when performing tasks that require high precision. A few other participants were also concerned about the helpers’ expertise: “it has to be very sophisticated to work with these [types] of complex circuits, and it has to be very intelligent for solving [these types] of uncommon issues or this type of issue as a firmware development software” (P8). These concerns suggest that future systems should offer information about the capabilities of the robot arm and the expertise of the helpers up front.

Three participants had concerns over whether they could trust the helper and their answers, a concern that can be allayed to an extent by presenting details of the helper’s background, expertise, and reputation. One of them (P2) suggested that “embodying the telepresence would probably be helpful. If there’s more authority behind that answering, and there’s more trustworthiness, it’d be helpful for me to trust it and use it.”

### **Wish for the robot to have automated functions**

Both during the study and interview, participants expected the robot arm to have some automated functions that did not rely on human experts. Participants reported that this expectation came from videos or articles that they previously read in which robots had more autonomy in performing tasks. Many commercial robots are independent and can perform the same tasks as the ones in our study, which explains why participants expected this.

*“It’d be nice to have an agent that’s like cleaning up after you’re done, and this just kind of does all the maintenance tasks that are [on the] circuit. (P8)”*

*“I have to pull 12 volt, 20 volt, 30 volt into my [circuits], and I have to do it manually. If I have anything that can do it automatically...it can save a lot of time for me. (P2)”*

## **6.4 Discussion**

In light of our findings, how can we design a system to make remote collaboration for embedded system development more effective? To address this question, we discuss four design implications in this section.

### **6.4.1 Providing “Beyond Being There” Technology**

Research has explored the development of effective remote collaboration techniques for nearly three decades [105, 108]. Prior work has recognized the importance of supporting “beyond being there” communication by developing techniques that offer more perceptual cues and physical affordance than face-to-face communication can [84]. Our study findings have shown that an expert-controlled robot as a conceptual intelligent assistant intrigued many potential “beyond being there” use cases, yet current off-the-shelf techniques provide insufficient guidance and support for either requesters or helpers to effectively leverage and benefit from the collaborative advantages of teleoperated robots and on-demand collaboration.

As Hollan and Storennetta advocated, there should be more attention paid to the needs that are not well met in unassisted face-to-face interaction [84]. From our observation, the participants' behaviors were driven by commonly used technologies, from their user interface (UI) design to the social norms of their users. The UIs of these technologies are inherited from versions designed to support conversations or digital information sharing, not physical tasks. The design principle used in these technologies focuses on allowing a more efficient exchange of information among collaborators, such as Google Doc's real-time document exchange and HP Halo's connection of two physically separated rooms through wall-size screens with high-fidelity audio and video. However, our study findings have shown that embedded system developers often need support for their physical artifacts via more clear guidance or physical aid, which are not well supported by current communication tools. This indicates a need for improving or re-designing support tools that consider the context of embedded system tasks. Specifically, we discuss a few design implications based on our findings and prior work on effective visual communication.

**Visual tools with privacy-sharing controls** To help requesters more easily set up and share their physical view and gestures, we suggest using a teleoperated, 360-degree camera that provides a wide-angle view. This technique could provide a maximized view of the physical environment for remote helpers, allowing them to see both the hardware and the requesters' communication cues (e.g., gestures, where they look). To enable helpers to operate remotely, future systems can adopt the target-tracking technique proposed by Rakita et al. [166], which uses a camera robot that automatically moves to provide a view with an appropriate camera-target distance.

Other than supporting real-time communication, the system could also record in 360-degree video for asynchronous communication. By teleoperating the camera (e.g., zooming, panning) or navigating the 360-degree video recording, remote helpers would be able to efficiently navigate around requesters' workspaces and perform tasks independently without interrupting requesters.

To reduce requesters' privacy concerns, prior work explored the privacy-utility tradeoff for remotely teleoperated and movable robots (e.g., iRobot) and found that blurring the view using a color-skewed superpixel filter was just as good as no filter regarding utility, but it offers more privacy [30]. Another survey revealed that responders were mostly concerned about shielding their personal and financial information from a household robot [14]. However, in the context of our study setting, where the participants conducted the embedded system development at



their workspaces, many tasks required very detailed contextual information (e.g., color), and none of the participants mentioned concerns about their financial or personal information being exposed to helpers in the workspace. Additionally, these prior studies focused on movable robots designed for household tasks; we instead looked at a fixed-position robot with lower mobility that was exclusively designed for embedded system collaboration. To address privacy concerns, we suggest that future systems could give requesters the ability to narrow the region of the camera view accessible to helpers (e.g., circuit board). Views outside the selected region would be blurred.

**Annotation tools for contextual explanation** To enable users to better pinpoint specific physical locations (e.g., a wire connection), we suggest that future support systems include object-tracking annotation tools that allow users to create virtual notes or marks. With this technique, users could sketch 3D annotations, highlight objects, or provide real-time guidance as the requester wires the circuits. In addition, the system could intelligently guide users to more easily integrate the responses from helpers by comparing old and updated circuits.

## 6.4.2 Adding Automation for High Precision and Easy Control for the Robot

A major challenge that many of our participants brought up was the quality of assistance that they would have received from the assistant. Part of this concern was from their prior experience interacting with teleoperated robot arms regarding their precision and flexibility. They worried that even with the right domain knowledge, helpers might perform poorly due to the maneuverability of the robot as well as their inexperience with robot arm control. We envision two directions for designing tools to facilitate precision and ease of control in teleoperated robot arms: autonomous assistance for teleoperation and routine task repetition.

**Autonomous Assistance for Teleoperation** Multiple studies have shown that adding autonomous motion and planning to a robot arm can help humans handle complex tasks. Leeper et al. studied how differences in the level of robot arm autonomy (i.e. a case in which the user needs to specify the target grasp position vs. one in which the user needs to click a pre-defined set of auto-computed grasp positions) affect a task's success rate. They found that people performed better when using the strategies with higher levels of autonomous assistance [122]. Kent et al. also compared two different user interfaces for remote robot arm control and suggested that the point-and-click user interface, which proposes grasp poses based on local 3D surface geometry,

can help users complete more tasks and reduce errors [96]. We suggest that future systems incorporate a certain level of autonomy into their robot arm to make it easy for remote helpers to operate.

**Routine Task Repetition for Requesters** In our study, requesters also wished that the robot agent had the ability to repeat certain tasks once the action was demonstrated. In fact, many industry collaborative robots (or cobots), such as uArm [193] or Universal Robotics cobots [171], have the ability to automate routine and repetitive processes in settings like manufacturing. However, prior work has found that users only used them because of their low cost and small size, rather than their flexible or collaborative characteristics [140]. With advances in machine learning, researchers have also proposed methods of having robots learn from humans after just a few demonstrations [70]. Based on requesters' perception of robot arms, we suggest future systems incorporate full automation into the robots for certain routine tasks, such as tuning input signals from a waveform generator, while still having human experts operate the arm for other tasks requiring flexibility and expertise (e.g., guiding the movements of certain wires or components on the circuits). In this way, helpers could even hand off more tasks to the robot arm itself, increasing overall productivity.

**Conditional Logic Control** Additionally, the system could also support if/else logic for various tasks. For example, the requester could ask the system to notify them if a measured signal is below a certain threshold or simply require it to keep displaying the signal.

### **6.4.3 Specialized Robot Arm for Embedded System Development Tasks**

With the combination of both software and hardware, a bulky robot arm can place an extra spatial burden on requesters. Two requesters were uncertain whether the robot arm could fit into their workspace given its size. On the other hand, three requesters proposed the integration of existing devices, such as measuring and displaying tools, into the arm to save space. To balance the concerns above with requesters' spatial needs, future support systems could integrate a few important functions into the robot arm. These could include a probe to measure different signals, as well as display and input/output devices for presenting information such as details on helpers' identities. The display could also serve as a resource for video chat communication, as well as a request list containing all unanswered requests, thus helping requesters manage their workflow.

#### **6.4.4 Identity, trust, expertise sharing, and capability**

While enhancing robot-mediated techniques can help address certain problems in remote collaboration, the solution to many other issues in this emerging device may exist fully or partially outside the purview of tools and technologies. Interpersonal trust has been shown to be radical for increasing cooperation in large organizations, especially when long-term relationships has not been established [164]. In the context of short-term collaboration with online crowds, requesters do not always know helpers' identities or levels of expertise. Prior work has shown that having a visual representation of the other person can improve information transfer and also increase development of trust over text-based communication [19]. Therefore, future systems should select those helpers who are willing to take such measures if the requester prefers to know their helpers' identities or wants a video chat prior to the start of the project.

In addition, the robot arm in our hypothetical study could introduce uncertainty among requesters, who may not understand the system's range of capabilities. A prior study showed that a robot capable of acting proactively is perceived as more helpful than one with no proactive function [160]. In the context of designing more helpful embedded system assistance, we suggest that future support systems proactively voice or present the capabilities of the system, including its precision, flexibility, and ability to verify requesters' demands. This proactivity should be adaptive based on the current tasks or the requesters' demands. For example, if a requester needs assistance with soldering, the system could inform them about its capacity to perform this task, including details on its precision.

### **6.5 Chapter Conclusion**

In this chapter, I explored the needs and challenges of supporting on-demand collaboration in embedded system development. By introducing the idea of using a teleoperated robot arm as a hypothetical assistant, our study provides insight into how and when developers choose to reach out for assistance and the challenges of supporting developers' needs via remote expert help. We have presented a set of features to address the primary needs that appeared in our study. In summary, this work informs, and brings into focus, a previously under-studied paradigm in remote embedded system collaboration.

# Chapter 7

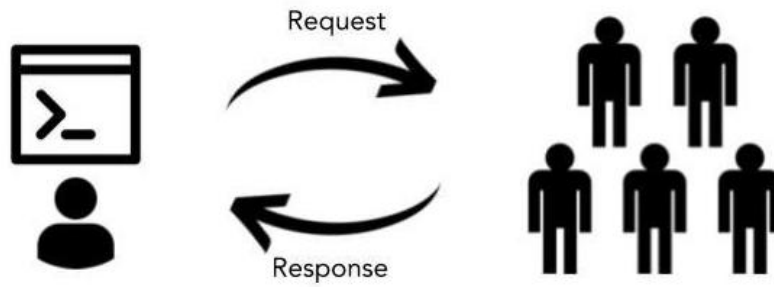
## Conclusion

This dissertation envisions a future in which developers of all kinds (e.g., software, hardware) can easily request on-demand assistance with necessary contextual information at any stage of their development process, and qualified agents (machine or human) can quickly be notified and coordinate their effort to promptly respond to the requests. By introducing and evaluating the idea of on-demand remote expert crowd assistance in software and embedded system development, I have demonstrated how developers can increase their productivity and better manage their workflow with this type of support system. I have also listed design suggestions for improving existing support systems and for future support systems or platform development.

### 7.1 Recap of Each Project

My thesis presents four projects to explore different aspects of this new class of on-demand expert support model. These aspects include needs, feasibility, benefits, and potential. As introduced in Chapter 1 (Figure 7.1), there are two main parts of this support model: coordination and communication. In this section, I reflect on how each of the four projects addresses the needs in these two categories.

The first two projects described in Chapter 3 (two studies) and Chapter 4 (CodeOn) discuss the motivation of this approach (e.g., challenges when using existing tools) and feasibility of this model (e.g., whether developers can benefit from the CodeOn system). In the CodeOn project, the series of small studies that I have conducted explored the design space of on-demand expert support systems regarding communication efficiency (i.e., speed and accuracy). By enabling both synchronous and asynchronous communication methods, I have also shown how this model



**Figure 7.1:** Diagram of the on-demand expert programming support paradigm.

can reduce coordination costs by allowing requester participants to parallelize their workflow, rather than relying on one communication method, while gaining efficiency.

Chapter 5 (CoCapture) extends this support model from code-related tasks to visual-program-output related tasks. It explores the communication challenge when the task is not symbolic and textual but visually interactive and dynamic (e.g., interactive user interface behavior). Chapter 6 further extends this model to embedded system development, exploring the communication and coordination costs that developers navigate when requesting on-demand remote expert support. These two projects have not only explored this new class of support system; compared to existing common practices, they also enable new methods of help-seeking interactions through direct manipulation.

## 7.2 Contributions

Overall, my thesis makes both empirical and technical contributions to the HCI community. These contributions fall into two large research areas: software development and help-seeking. In this section, I will reflect on lessons learned through my dissertation in these categories.

### Software Development

Compared to on-demand help services on the market such as Uber and Task Rabbit, I have shown that on-demand collaboration in software development requires much more context exchange between requesters and helpers. Helpers also need more controlled access to requesters' work-related materials (e.g., codebase). This results from the nature of software development—a highly sophisticated task requires strong logical skill, memorization of code syntax, and years of practice. Because of the complexity, the developers face various challenges when creating a task,

such as how to formulate the task for someone who does not have prior knowledge of the project, and what context to include in the task to make it easy to understand. Unlike those dedicated services which mostly focus on micro-level tasks (e.g., running errands), these challenges can lead to programming tasks of different scales. With different scales, helpers' expertise, the needed time and cost to complete the task, and requesters' expectations of the output can vary.

This dissertation focuses on increasing developers' productivity in software development, including the number of questions or tasks that they can resolve. The developers include both requesters and helpers. One of the challenges in increasing productivity is coordination; as Fred Brooks claimed in his book *The Mythical Man-Month: Essays on Software Engineering*, "adding manpower to a late software project makes it later."

My thesis has shed a light on the question of how to help developers better manage their workflow when on-demand expert support is available. This includes fewer interruptions and more flexibility for requesters when navigating between their own tasks and helpers' responses. On the helper side, I have also studied methods of better coordinating users to increase overall productivity. One project that I did not discuss in this dissertation explored two techniques, interactive event-flow graphs and GUI-level guidance, which guide GUI testers (helpers) to discover more test cases and avoid duplicate test cases [44]. These techniques and findings can help guide coordination among helpers in other domains.

## **Help-Seeking**

My dissertation focuses on the communication and representation of information during help-seeking. This includes the process of requesters making a request, the way in which helpers see this information, and how they respond to requesters in ways that are more appropriate and expected. My thesis has shown that helpers often need further clarification about complex tasks that they may not need for simpler microtasks. Reducing communication delay, such as the number of times people must communicate back and forth, is one of the essential goals considered throughout this thesis. As I have shown, the on-demand help-seeking model proposed in this dissertation can reduce communication costs by enabling requesters to create more comprehensive coding tasks and helpers to provide easily integrated responses. This in-context communication reduces the effort of creating a proxy (e.g., using natural language) to describe the information.

Beyond help-seeking in software development, the three stages of the help-seeking process

discussed earlier—making a request, writing a response, and integrating the response—fit into a more general help-seeking context. Specifically, tools for generating sub-tasks in the current context of work, aiding helpers in “rehydrating” that context, and facilitating quick and effective integration of contributions can be used across various domains, such as using software made for professionals (e.g. Photoshop or Final Cut Pro). Exploring how to effectively recreate this approach in other settings is interesting for future work.

While increasing productivity is an important goal in many help-seeking settings, learning is another common goal requesters have when reaching out for support. During my Ph.D., I have also conducted a few other projects that looked at learning outcomes in programming [43]. Collaboration in software development is distinct from collaborating in an educational setting. For instance, learners often lack the knowledge and understanding needed to correctly phrase a question when compared to professional programmers. Instructors often prefer to guide students with hints and questions rather than giving away answers, whereas professional programmers tend to offer straightforward solutions that can be replicated by others. Additionally, instructional support can be hard to scale in courses that have high student-teacher ratios. Furthermore, this instructional support often suffers from an expert blind spot that limits recognition of how students will misunderstand explanations [145].

As an initial step in the space of programming education, I conducted two studies to better understand the trade-off between personalization and scalability for support in programming education. To do this, I used CodeOn, a tool shown to improve the productivity of software programmers. By carefully designing new techniques, I created EdCode, which extends CodeOn and helps scale personalized support in an educational setting. In the future, I am excited about conducting similar studies that make contributions in other areas.

## 7.2.1 Summary of Contributions

- *Crowd computing in software development.* We motivated a new support paradigm—on-demand remote expert support—for software development to address often delayed and ineffective help-seeking interactions. Through two studies, we found that software developers want to have support systems built around this paradigm, and we also showed that state-of-the-art techniques are insufficient to adequately instantiate this paradigm. Analyzing these two studies, we provided deep understanding and evidence that could support future system design with this new paradigm.

- *Communication and coordination system design.* This thesis also designed, implemented, and studied a series of interactive techniques, including two systems, to help users to make clear requests, organize their tasks, and/or receive easily understood and integrated responses. Specifically, CodeOn and CoCapture are two programming support systems that enable software developers and UI designers to receive expert crowd assistance for tasks that are related to code and UI behaviors, respectively. Our mixed-method case comparison revealed the limitations of traditional synchronous communication tools and identified several key elements missing from the on-demand support systems. It also highlights the advantages of on-demand remote expert support paradigms, which could coordinate the collective effort of experts and non-experts at scale.
- *Remote collaboration in embedded system development.* Beyond software development, this thesis also explores embedded system development, which often involves both software and hardware development. Although contextual information is necessary during collaboration, we found that system developers still rely on basic support tools for remote collaboration. Through a study using a conceptual prop and literature review across fields of human-robotic interaction, software development, and remote collaborative work for physical tasks, this thesis identified various challenges of remote collaboration in embedded system development using existing techniques and contributed a list of design recommendations for future system designers.

### **7.3 Novelty of the Work**

In my dissertation work, I designed and implemented a set of novel interactive techniques that have been shown to be effective in addressing these challenges in software development collaboration. The core novelty behind these techniques is that they capture requesters' task/question creation process, thus providing richer contextual dynamic, temporal, and spatial information. For example, these techniques capture the dynamic behavior of the code and program output (e.g., UI behavior), the relevant code snippets from multiple files, or historical information such as requesters' prior attempts to fix their bugs. Unlike video recording or screenshot media, whose output captures only audio and visual information, the software developed in my thesis



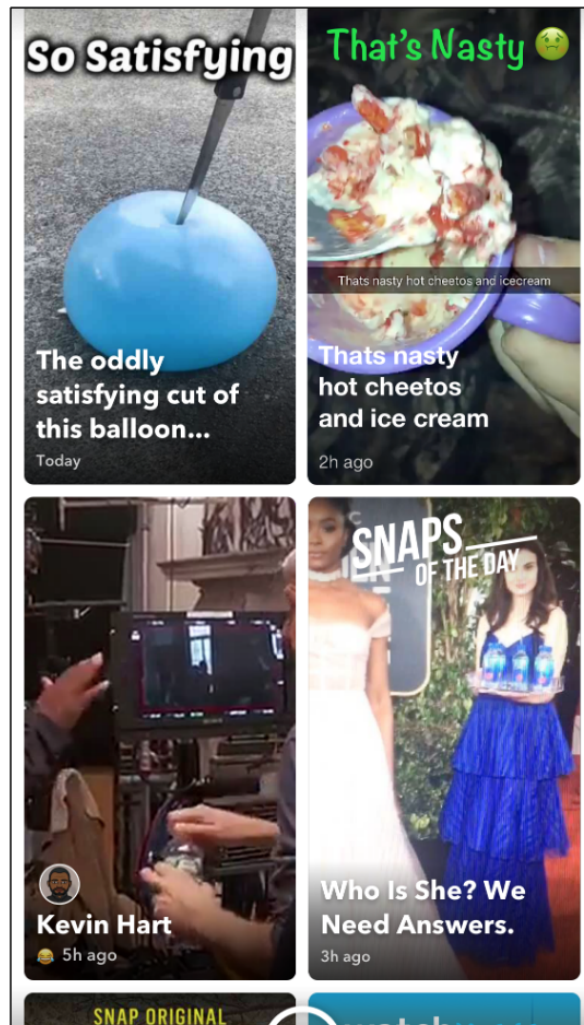
work allows users to create element-based recordings. These recordings allow them to capture, manipulate, and remix their interactions with an application. In this way, requesters can adjust their task creation process and helpers can directly build on top of the element-based recording. Overall, these methods reduce communication delay by making the task creation process easier, the task descriptions clearer and more accurate, and helpers' responses easy to integrate into requesters' codebases.

## 7.4 Benefits Beyond Software Development

This dissertation has demonstrated the effectiveness of the on-demand help-seeking model in software development. The target audiences are programmers who need support that can be more efficiently provided by remote expert developers than existing methods. Aside from this target audience group and the collaboration goal of increased productivity, this remote assistance model can be useful in many contexts, such as education and video analysis. In this section, I will briefly describe a few projects I have also worked on during my Ph.D. that used this help-seeking model in other contexts.

In educational settings, providing personalized support to students at scale has always been challenging. Many courses use Q&A forums (e.g., Piazza) to scale support for students, but forum participation is often low due to their public nature and the inability to have a natural conversation [23, 133, 162]. Several systems, including CodeOn [42], have been proven effective at scaling remote programming assistance in work settings. However, programming support in educational settings is different from that of work settings in terms of goals, stakeholders' expertise, and collaboration structure [37]. For example, students often lack the knowledge and understanding needed to phrase a question correctly compared to professional programmers. Instructors would prefer to guide students with hints and questions rather than giving away answers, whereas professional programmers tend to offer straightforward solutions that can be replicated by others.

To address these challenges, I have developed EdCode [43], a variant of CodeOn, as a remote support system that allows instructors to provide personalized assistance to students and publicly share their support with coding questions in programming classes. EdCode does this by adding three new features on top of CodeOn: **contextualized explanations**, which allow students to make text-based help requests with code context (by highlighting relevant code snippets) from



**Figure 7.2:** Screenshot from 2018 of some curated content from users’ posts in Snapchat, e.g., “So Satisfying: The oddly satisfying cut of this balloon...”

their working context (IDE state) in asynchronous fashion; a **chat tool** that allows instructors to converse with students within each question; and **selective code publicity**, which allows instructors to publish students’ questions and answers by selecting code regions that are relevant to the question while concealing the rest of the solution. In this way, other students can review questions and answers curated by the instructors, leading to fewer duplicate questions and saving time for instructors and students alike.

I conducted a virtual office hour study in an introductory programming course. Both student and instructor participants reported that the quality of answers from EdCode was better than their existing Q&A forum (a Facebook Group). Students also found that the answers were comparable to those obtained from in-person office hours, and they could understand the questions and answers with only the curated version of the original code visible.

Outside of programming, I have also explored how to provide efficient on-demand assistance in video curation, which is also a type of task that requires much context information and a wide range of expertise to perform well. In particular, I studied the video curation task on Snapchat, a user-generated content platform. Snapchat uses a curator-based approach that relies on staff curators who identify and organize compelling content before posting it to their platform for further user consumption [178] (Fig. 7.2). This approach gives platforms editorial control and overcomes machines' inability to make subjective assessments and prevent adversarial users from manipulating content selection, but it is limited in terms of scale [53]. Specifically, it is difficult to scale curators' ability to find appropriate content from a corpus of videos that is large and rapidly growing—on YouTube, for example, over 500 hours of video content is uploaded every minute.

To overcome this limitation, I designed and implemented Sifter [45], a system that improves the curation process by combining automated techniques with a human-powered pipeline that browses, selects, and reaches an agreement on what videos to include in a compilation. I evaluated Sifter by creating 12 compilations from over 34,000 user-generated videos. Sifter was more than three times faster than dedicated curators, and its output was of comparable quality. The promising results in this project have further confirmed the effectiveness and applicability of this on-demand help-seeking model for complex tasks.

## 7.5 The Future of Work

Towards the end of my Ph.D. journey, the coronavirus pandemic significantly disrupted information work across the globe. Technology companies, schools, bootcamps, and makerspaces have shifted to prolonged online work as a result of COVID-19. This change in working environment directly affects one of the most common human behaviors at work—asking for support from others during program development. With programmers working from a relaxed environment at home, help-seeking behavior has shifted from synchronous and in-person to asynchronous and remote. Although more flexible in terms of location and time, overhead coordination costs and ineffective communication (e.g., insufficient context) often delay the interaction, reducing overall productivity.

I envision a new collaboration model in which developers can easily make comprehensive on-demand requests that include appropriate context and remote helpers can quickly provide

appropriate and high-quality assistance. This will enable more personalized question-answering or task hand-off types of assistance and more efficient allocation of expertise than current techniques. I also advocate facilitating effective context capturing and workforce coordination to help scale the current remote work situation [120]. By working toward this vision, we will better understand what questions developers need answered remotely, how they ask them, and how the existing question-answering systems can support their requests. It will enable new types of programming collaboration and teamwork by using a combination of human and machine intelligence to complete programming tasks.

This dissertation made initial steps to fulfill this vision. As I have shown, the produced tools and techniques in my work can make developers more efficient and thus more productive overall. The design implications could provide guidance for future system designers to build better support tools that allow software and embedded system developers to receive on-demand assistance. The systems are the first to enable developers to hand off request-based physical computing tasks on demand without relinquishing control of the devices.

In the long run, the resulting techniques can also help expand participation in physical computing design and development in educational and collaborative work as well as DIY settings. Since shifting to an online setting, for example, programming courses across the country have had to adjust the difficulty level of the coursework or even reduce the number of collaborative projects to allow for remote study. I envision a future in which students can more easily collaborate and instructors can more easily track and facilitate their progress through resulting tools. With more efficient remote support, instructors can host virtual office hours instead of in-person hours and more easily understand comprehension gaps by viewing learners' past attempts, thus reducing the interaction delay. Additionally, the produced tools could enable more collaboration, from peer feedback to team projects, than digital-only communication techniques between learners, thus helping to increase learner engagement.

In particular, I outline two directions that I want to continue exploring in the future.

- *Automated Techniques for Programming Assistance.* This thesis has focused on using human helpers to provide programmers with programming support. One problem with this approach is that the helpers often have to write their answers from scratch. Much existing work (e.g., <https://kite.com/>) has attempted to use machine intelligence to provide programming support, yet they are limited to a small number of use cases. Two challenges

for understanding and performing programming tasks are that 1) they often require domain knowledge, and 2) they often require additional context. Future work can explore methods to improve existing AI models by using humans to contribute to various parts of the pipeline, such as collecting more useful training data at a low cost.

- *On-demand Collaboration in a Programming Learning Environment.* This thesis has shown that by creating specific tools to address the challenges people face during on-demand collaboration, we can increase the productivity of software developers. However, if the goal is to help people better teach and learn about software development, what challenges would they face during on-demand collaboration? Would systems like CodeOn still be effective in helping them achieve their goal? How might we address any new challenges, and what can we borrow from previous findings? Future work can explore these questions by exploring the goals, challenges, and needs people have during on-demand collaboration in programming learning settings.

Taken together, this thesis re-imagines a future of work in which developers with different levels of expertise no longer rely on outdated collaboration tools. Instead, they effortlessly request assistance for any kind of development tasks at any time, relying on qualified helpers to provide trustworthy and timely support. The result of this vision is an intelligent system that empowers anyone to coordinate and scale their efforts with machine and human agents in pursuit of complex, open-ended goals.

# Bibliography

- [1] M. S. Ackerman and T. W. Malone. “Answer Garden: A Tool for Growing Organizational Memory”. In: *SIGOIS Bull.* 11.2–3 (Mar. 1990), pp. 31–39. ISSN: 0894-0819. DOI: [10.1145/91478.91485](https://doi.org/10.1145/91478.91485). URL: <https://doi.org/10.1145/91478.91485>.
- [2] Mark S Ackerman. “Augmenting organizational memory: a field study of answer garden”. In: *ACM Transactions on Information Systems (TOIS)* 16.3 (1998), pp. 203–224.
- [3] Mark S Ackerman and David W McDonald. “Answer Garden 2: merging organizational memory with collaborative help”. In: *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. ACM. 1996, pp. 97–105.
- [4] “Adobe XD”. In: Accessed: April, 2020. URL: <https://www.adobe.com/products/xd.html>.
- [5] Luis Afonso et al. “Effect of hand-avatar in a selection task using a tablet as input device in an immersive virtual environment”. In: *3D User Interfaces*. 2017.
- [6] Leila Alem and Jane Li. “A study of gestures in a video-mediated collaborative assembly task”. In: *Advances in Human-Computer Interaction 2011* (2011), p. 1.
- [7] Leila Alem, Franco Tecchia, and Weidong Huang. “HandsOnVideo: Towards a gesture based mobile AR system for remote collaboration”. In: *Recent trends of mobile collaborative augmented reality systems*. Springer, 2011.
- [8] Abdullah Almaatouq et al. “The influence of social norms on synchronous versus asynchronous communication technologies”. In: *Proceedings of the 1st ACM international workshop on Personal data meets distributed multimedia*. ACM. 2013, pp. 39–42.
- [9] *Amazon Inc.*, <https://www.mturk.com>. Accessed: Dec., 2019. 2015. URL: <https://www.mturk.com>.

- [10] Judith Amores, Xavier Benavides, and Pattie Maes. “Showme: A remote collaboration system that supports immersive gestural communication”. In: *CHI*. 2015.
- [11] Muhammad Asaduzzaman et al. “Answering questions about unanswered questions of stack overflow”. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press. 2013, pp. 97–100.
- [12] Ignacio Avellino et al. “Multimodal and Mixed Control of Robotic Endoscopes”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–14.
- [13] Prashant Baheti, Edward Gehringer, and David Stotts. “Exploring the efficacy of distributed pair programming”. In: *Extreme Programming and Agile Methods—XP/Agile Universe 2002*. Springer, 2002, pp. 208–220.
- [14] Sogol Balali et al. “Privacy Concerns in Robot Teleoperation: Does Personality Influence What Should Be Hidden?” In: *International Conference on Social Robotics*. Springer. 2019, pp. 719–729.
- [15] Xavier Benavides, Judith Amores, and Pattie Maes. “Remot-IO: a System for Reaching into the Environment of a Remote Collaborator”. In: *UIST*. 2015.
- [16] Michael S Bernstein et al. “Crowds in two seconds: Enabling realtime crowd-powered interfaces”. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM. 2011, pp. 33–42.
- [17] Jeffrey P Bigham et al. “VizWiz: nearly real-time answers to visual questions”. In: *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. ACM. 2010, pp. 333–342.
- [18] Tracey Booth et al. “Crossed wires: Investigating the problems of end-user developers in a physical computing task”. In: *CHI*. 2016.
- [19] Elizabeth A Boyle, Anne H Anderson, and Alison Newlands. “The effects of visibility on dialogue and performance in a cooperative problem solving task”. In: *Language and speech* 37.1 (1994), pp. 1–20.
- [20] Joel Brandt et al. “Example-centric programming: integrating web search into the development environment”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 513–522.

- [21] Joel Brandt et al. “Two studies of opportunistic programming: interleaving web foraging, learning, and writing code”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2009, pp. 1589–1598.
- [22] Joel Brandt et al. “Writing Code to Prototype, Ideate, and Discover”. In: *Software, IEEE* 26.5 (2009), pp. 18–24.
- [23] Lori Breslow et al. “Studying learning in the worldwide classroom: Research into edX’s first MOOC”. In: *Research & Practice in Assessment* 8 (2013).
- [24] Frederick P Brooks. *The mythical man-month*. Vol. 1995. Addison-Wesley Reading, MA, 1975.
- [25] Jed R Brubaker, Gina Venolia, and John C Tang. “Focusing on shared experiences: moving beyond the camera in video communication”. In: *Proceedings of the Designing Interactive Systems Conference*. 2012, pp. 96–105.
- [26] Brian Burg, Amy J Ko, and Michael D Ernst. “Explaining visual changes in web interfaces”. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM. 2015, pp. 259–268.
- [27] Brian Burg et al. “Interactive record/replay for web application debugging”. In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 2013, pp. 473–484.
- [28] N. Burtnyk and M. Wein. “Computer-Generated Key-Frame Animation”. In: *Journal of the SMPTE* 80 (1971), pp. 149–153.
- [29] Daniel J. Butler et al. “The Privacy-Utility Tradeoff for Remotely Teleoperated Robots”. In: *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. HRI ’15. Portland, Oregon, USA: Association for Computing Machinery, 2015, pp. 27–34. ISBN: 9781450328838.
- [30] Daniel J Butler et al. “The privacy-utility tradeoff for remotely teleoperated robots”. In: *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*. 2015, pp. 27–34.
- [31] Kerry Shih-Ping Chang and Brad A Myers. “WebCrystal: understanding and reusing examples in web authoring”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2012, pp. 3205–3214.



- [32] Yun Suk Chang et al. “Evaluating gesture-based augmented reality annotation”. In: *3D User Interfaces (3DUI), 2017 IEEE Symposium on*. IEEE. 2017, pp. 182–185.
- [33] Kathy Charmaz. *Constructing grounded theory: A practical guide through qualitative analysis*. sage, 2006.
- [34] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. “Rousillon: Scraping Distributed Hierarchical Web Data”. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 2018, pp. 963–975.
- [35] Jeff Chastine et al. “Studies on the effectiveness of virtual pointers in collaborative augmented reality”. In: *3D User Interfaces*. IEEE. 2008.
- [36] Yan Chen, Jasmine Jones, and Yaxing Yao. “WireOn: Supporting Remote Collaboration for Embedded System Development”. In: *Conference Companion Publication of the 2020 on Computer Supported Cooperative Work and Social Computing*. 2020, pp. 7–11.
- [37] Yan Chen, Walter S Lasecki, and Tao Dong. “Towards Supporting Programming Education at Scale via Live Streaming”. In: *arXiv preprint arXiv:2010.15015* (2020).
- [38] Yan Chen, Sang Won Lee, and Steve Oney. “CoCapture: Effectively Communicating UI Behaviors on Existing Websites by Demonstrating and Remixing”. In: Under review.
- [39] Yan Chen, Steve Oney, and Walter Lasecki. “Expert Crowd Support Systems for Software Developers”. In: CI’16.
- [40] Yan Chen, Steve Oney, and Walter S Lasecki. “Towards providing on-demand expert support for software developers”. In: *Proceedings of the 2016 CHI conference on human factors in computing systems*. ACM. 2016.
- [41] Yan Chen et al. “Bashon: A Hybrid Crowd-Machine Workflow for Shell Command Synthesis”. In: *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2020, pp. 1–8.
- [42] Yan Chen et al. “Codeon: On-demand software development assistance”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM. 2017.
- [43] Yan Chen et al. “EdCode: Towards Personalized Support at Scale for Remote Assistance in CS Education”. In: *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2020, pp. 1–5.

- [44] Yan Chen et al. “Improving Crowd-Supported GUI Testing with Structural Guidance”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–13.
- [45] Yan Chen et al. “Sifter: A Hybrid Workflow for Theme-based Video Curation at Scale”. In: *ACM International Conference on Interactive Media Experiences*. 2020, pp. 65–73.
- [46] Pei-Yu Chi, Sen-Po Hu, and Yang Li. “Doppio: Tracking ui flows and code changes for app development”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–13.
- [47] Parmit K Chilana et al. “Supporting Remote Real-Time Expert Help: Opportunities and Challenges for Novice 3D Modelers”. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2018, pp. 157–166.
- [48] Jan Chong and Tom Hurlbutt. “The social dynamics of pair programming”. In: *29th International Conference on Software Engineering (ICSE’07)*. IEEE. 2007, pp. 354–363.
- [49] Herbert H Clark and Susan E Brennan. “Grounding in communication.” In: (1991).
- [50] Alistair Cockburn and Laurie Williams. “The costs and benefits of pair programming”. In: *Extreme programming examined* (2000), pp. 223–247.
- [51] Juliet Corbin, Anselm L Strauss, and Anselm Strauss. *Basics of qualitative research*. sage, 2015.
- [52] Cecilia Kremer Vieira da Cunha and Clarisse Sieckenius de Souza. “Toward a culture of end-user programming understanding communication about extending applications”. In: 2003.
- [53] Edward Curry, Andre Freitas, and Sean O’Riáin. “The Role of Community-Driven Data Curation for Enterprises”. In: *Linking Enterprise Data*. Ed. by David Wood. Boston, MA, 2010.
- [54] JY Didier et al. “AMRA: augmented reality assistance in train maintenance tasks. The 4th ACM”. In: *ISMAR*. 2005.
- [55] Paul Dourish and Victoria Bellotti. “Awareness and coordination in shared workspaces”. In: *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. 1992, pp. 107–114.

- [56] Omid Fakourfar et al. “Stabilized annotations for mobile remote assistance”. In: *CHI*. ACM. 2016.
- [57] “Figma”. In: Accessed: March, 2020. <https://www.figma.com/>. 2020.
- [58] Susan R Fussell, Robert E Kraut, and Jane Siegel. “Coordination of communication: Effects of shared visual context on collaborative work”. In: *CSCW*. 2000.
- [59] Susan R Fussell, Leslie D Setlock, and Robert E Kraut. “Effects of head-mounted and scene-oriented video systems on remote collaboration on physical tasks”. In: *CHI*. 2003.
- [60] Susan R Fussell et al. “Gestures over video streams to support remote collaboration on physical tasks”. In: *Human-Computer Interaction* 19.3 (2004), pp. 273–309.
- [61] Lei Gao et al. “An oriented point-cloud view for MR remote collaboration”. In: *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications*. ACM. 2016, p. 8.
- [62] Lei Gao et al. “Static local environment capturing and sharing for MR remote collaboration”. In: *SIGGRAPH Asia Mobile Graphics & Interactive Applications*. 2017.
- [63] Steffen Gauglitz et al. “In touch with the remote world: Remote collaboration with augmented reality drawings and virtual navigation”. In: *VRST*. 2014.
- [64] Steffen Gauglitz et al. “Integrating the physical environment into mobile remote collaboration”. In: *MobileHCI*. 2012.
- [65] Steffen Gauglitz et al. “World-stabilized annotations and virtual scene navigation for remote collaboration”. In: *UIST*. 2014.
- [66] Darren Gergle, Robert E Kraut, and Susan R Fussell. “Using visual information for grounding and awareness in collaborative tasks”. In: *Human-Computer Interaction* 28.1 (2013), pp. 1–39.
- [67] Max Goldman, Greg Little, and Robert C Miller. “Real-time collaborative coding in a web IDE”. In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM. 2011, pp. 155–164.
- [68] “Google Inc.” In: Accessed: March, 2020. <https://github.com/GoogleChromeLabs/ProjectVisBug>. 2020.
- [69] Lars Grammel, Melanie Tory, and Margaret-Anne Storey. “How information visualization novices construct visualizations”. In: *IEEE transactions on visualization and computer graphics* 16.6 (2010), pp. 943–952.

- [70] Daniel H Grollman and Odest Chadwicke Jenkins. “Incremental learning of subtasks from unsegmented demonstration”. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 261–266.
- [71] Tovi Grossman, Justin Matejka, and George Fitzmaurice. “Chronicle: capture, exploration, and playback of document workflow histories”. In: *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. 2010, pp. 143–152.
- [72] Philip J Guo. “Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming”. In: *Proceedings of the 28th annual ACM symposium on User interface software and technology*. ACM. 2015.
- [73] Philip J Guo, Jeffery White, and Renan Zanelatto. “Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning”. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*. IEEE. 2015.
- [74] Pavel Gurevich, Joel Lanir, and Benjamin Cohen. “Design and implementation of teleadvisor: a projection-based augmented reality system for remote collaboration”. In: *Computer Supported Cooperative Work (CSCW) (2015)*.
- [75] Carl Gutwin and Saul Greenberg. “A descriptive framework of workspace awareness for real-time groupware”. In: *Computer Supported Cooperative Work (CSCW) 11.3-4 (2002)*, pp. 411–446.
- [76] Carl Gutwin, Mark Roseman, and Saul Greenberg. “A usability study of awareness widgets in a shared workspace groupware system”. In: *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. 1996, pp. 258–267.
- [77] Carl Gutwin, Gwen Stark, and Saul Greenberg. “Support for workspace awareness in educational groupware.” In: *CSCL*. Vol. 95. 1995, pp. 147–156.
- [78] Anja Guzzi et al. “Supporting Developers’ Coordination in the IDE”. In: *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM. 2015, pp. 518–532.
- [79] Björn Hartmann et al. “What would other programmers do: suggesting solutions to error messages”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2010, pp. 1019–1028.

- [80] Steven J Henderson and Steven K Feiner. “Augmented reality in the psychomotor phase of a procedural task”. In: *ISMAR*. 2011.
- [81] James D Herbsleb et al. “Object-oriented analysis and design in software project teams”. In: *Human–Computer Interaction* 10.2-3 (1995), pp. 249–292.
- [82] Joshua Hibsichman and Haoqi Zhang. “Telescope: Fine-tuned discovery of interactive web UI feature implementation”. In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 2016, pp. 233–245.
- [83] Joshua Hibsichman and Haoqi Zhang. “Unravel: Rapid web application reverse engineering via interaction recording, source tracing, and library detection”. In: *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM. 2015, pp. 270–279.
- [84] Jim Hollan and Scott Stornetta. “Beyond being there”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1992, pp. 119–125.
- [85] Damon Horowitz and Sepandar D Kamvar. “The anatomy of a large-scale social search engine”. In: *Proceedings of the 19th international conference on World wide web*. ACM. 2010, pp. 431–440.
- [86] Weidong Huang and Leila Alem. “HandsinAir: a wearable system for remote collaboration on physical tasks”. In: *CSCW*. 2013.
- [87] Weidong Huang, Leila Alem, and Franco Tecchia. “HandsIn3D: supporting remote guidance with immersive virtual environments”. In: *IFIP Conference on Human-Computer Interaction*. Springer. 2013, pp. 70–77.
- [88] Alexander Hubers et al. “Using video manipulation to protect privacy in remote presence systems”. In: *International Conference on Social Robotics*. Springer. 2015, pp. 245–254.
- [89] Cloud9 IDE Inc. *Cloud9 IDE*. Accessed: September, 2015. 2010. URL: <https://c9.io>.
- [90] Codementor Inc. *Code Mentor*. Accessed: September, 2015. 2014. URL: <https://codementor.io/>.
- [91] Hiwonder Inc. Accessed: July, 2020. 2020. URL: <https://www.hiwonder.hk/collections/robotic-arm>.
- [92] Pluralsight Inc. *hack.hands()*. Accessed: September, 2015. 2013. URL: <https://hackhands.com/>.

- [93] Kori Inkpen et al. “Experiences2Go: sharing kids’ activities outside the home with remote family members”. In: *Proceedings of the 2013 conference on Computer supported cooperative work*. 2013, pp. 1329–1340.
- [94] Shamsi T Iqbal and Eric Horvitz. “Disruption and recovery of computing tasks: field study, analysis, and directions”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2007, pp. 677–686.
- [95] Mitchell Karchemsky et al. “Heimdall: A Remotely Controlled Inspection Workbench For Debugging Microcontroller Projects”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12.
- [96] David Kent, Carl Saldanha, and Sonia Chernova. “Leveraging depth data in remote robot teleoperation interfaces for general object manipulation”. In: *The International Journal of Robotics Research* 39.1 (2020), pp. 39–53.
- [97] Seungwon Kim, Gun A Lee, and Nobuchika Sakata. “Comparing pointing and drawing for remote collaboration”. In: *ISMAR*. 2013.
- [98] David S Kirk and Danaë Stanton Fraser. “The effects of remote gesturing on distance instruction”. In: (2005).
- [99] David Kirk, Tom Rodden, and Danaë Stanton Fraser. “Turn it this way: grounding collaborative action with remote gestures”. In: *CHI*. ACM. 2007.
- [100] David Kirk and Danae Stanton Fraser. “Comparing remote gesture technologies for supporting collaborative physical tasks”. In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006, pp. 1191–1200.
- [101] Jeffrey Klow et al. “Privacy, utility, and cognitive load in remote presence systems”. In: *International Conference on Social Robotics*. Springer. 2019, pp. 730–739.
- [102] Amy J Ko, Robert DeLine, and Gina Venolia. “Information needs in collocated software development teams”. In: *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society. 2007, pp. 344–353.
- [103] Amy J Ko, Brad Myers, Htet Htet Aung, et al. “Six learning barriers in end-user programming systems”. In: *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE. 2004, pp. 199–206.

- [104] Kenneth R Koedinger and Mitchell J Nathan. “The real story behind story problems: Effects of representations on quantitative reasoning”. In: *The journal of the learning sciences* 13.2 (2004), pp. 129–164.
- [105] Robert M Krauss and Susan R Fussell. “Mutual knowledge and communicative effectiveness”. In: *Intellectual teamwork: Social and technological foundations of cooperative work* (1990), pp. 111–146.
- [106] Robert E Kraut, Susan R Fussell, and Jane Siegel. “Visual information as a conversational resource in collaborative physical tasks”. In: *Human–computer interaction* 18.1-2 (2003), pp. 13–49.
- [107] Robert E Kraut, Darren Gergle, and Susan R Fussell. “The use of visual information in shared visual spaces: Informing the development of virtual co-presence”. In: *CSCW*. 2002.
- [108] Robert E Kraut, Mark D Miller, and Jane Siegel. “Collaboration in performance of physical tasks: Effects on outcomes and communication”. In: *Proceedings of the 1996 ACM conference on Computer supported cooperative work*. 1996, pp. 57–66.
- [109] Rebecca Krosnick et al. “Expresso: Building responsive interfaces with keyframes”. In: *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2018, pp. 39–47.
- [110] Hideaki Kuzuoka. “Spatial workspace collaboration: a SharedView video support system for remote collaboration capability”. In: *CHI*. 1992.
- [111] Hideaki Kuzuoka, Toshio Kosuge, and Masatomo Tanaka. “GestureCam: A video communication system for sympathetic remote collaboration”. In: *CSCW*. 1994.
- [112] James A Landay and Brad A Myers. “Interactive sketching for the early stages of user interface design”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1995, pp. 43–50.
- [113] Walter S Lasecki et al. “Apparition: Crowdsourced user interfaces that come to life as you sketch them”. In: *CHI’15*.
- [114] Walter S Lasecki et al. “Chorus: a crowd-powered conversational assistant”. In: *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM. 2013, pp. 151–162.

- [115] Walter S Lasecki et al. “Glance: Rapidly coding behavioral video with the crowd”. In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM. 2014, pp. 551–562.
- [116] Walter Lasecki et al. “Real-time captioning by groups of non-experts”. In: *Proceedings of the 25th annual ACM symposium on User interface software and technology*. ACM. 2012, pp. 23–34.
- [117] Thomas D LaToza et al. “Microtask programming: Building software with a crowd”. In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM. 2014, pp. 43–54.
- [118] Gun A Lee et al. “Mixed reality collaboration through sharing a live panorama”. In: *SIGGRAPH Asia 2017 Mobile Graphics & Interactive Applications*. ACM. 2017, p. 14.
- [119] Sang Won Lee and Jason Freeman. “Real-time music notation in mixed laptop–acoustic ensembles”. In: *Computer Music Journal* 37.4 (2013), pp. 24–36.
- [120] Sang Won Lee et al. “Exploring coordination models for ad hoc programming teams”. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 2017, pp. 2738–2745.
- [121] Sang Won Lee et al. “Sketchexpress: Remixing animations for more effective crowd-powered prototyping of interactive interfaces”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM. 2017, pp. 817–828.
- [122] Adam Eric Leeper et al. “Strategies for human-in-the-loop robotic grasping”. In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. 2012, pp. 1–8.
- [123] Germán Leiva and Michel Beaudouin-Lafon. “Montage: A Video Prototyping System to Reduce Re-Shooting and Increase Re-Usability”. In: *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 2018, pp. 675–682.
- [124] Germán Leiva et al. “Pronto: Rapid Augmented Reality Video Prototyping Using Sketches and Enaction”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2020.



- [125] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. “SUGILITE: creating multimodal smartphone automation by demonstration”. In: *Proceedings of the 2017 CHI conference on human factors in computing systems*. 2017, pp. 6038–6049.
- [126] Yang Li et al. “FrameWire: a tool for automatically extracting interaction logic from paper prototyping tests”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2010, pp. 503–512.
- [127] James Lin et al. “DENIM: finding a tighter fit between tools and practice for Web site design”. In: *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 2000, pp. 510–517.
- [128] Greg Little et al. “Turkit: human computation algorithms on mechanical turk”. In: *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. ACM. 2010, pp. 57–66.
- [129] Xiaoyong Liu, W Bruce Croft, and Matthew Koll. “Finding experts in community-based question-answering services”. In: *Proceedings of the 14th ACM international conference on Information and knowledge management*. 2005, pp. 315–316.
- [130] Rajan M Lukose et al. “Shock: communicating with computational messages and automatic private profiles”. In: *Proceedings of the 12th international conference on World Wide Web*. ACM. 2003, pp. 291–300.
- [131] Lambros Makris et al. “Teleworks: A CSCW application for remote medical diagnosis support and teleconsultation”. In: *IEEE transactions on information technology in biomedicine* (1998).
- [132] Lena Mamykina et al. “Design lessons from the fastest q&a site in the west”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM. 2011, pp. 2857–2866.
- [133] J Manning and M Sanders. “How widely used are MOOC forums? A first look”. In: *Signals: Thoughts on Online Learning* (2013).
- [134] Michael McCurdy et al. “Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success”. In: *Proceedings of the SIGCHI conference on Human Factors in computing systems*. 2006, pp. 1233–1242.

- [135] David W McDonald and Mark S Ackerman. “Expertise recommender: a flexible recommendation system and architecture”. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM. 2000, pp. 231–240.
- [136] David W McDonald, Chunhua Weng, and John H Gennari. “The multiple views of inter-organizational authoring”. In: *Proceedings of the 2004 ACM conference on Computer supported cooperative work*. 2004, pp. 564–573.
- [137] Will McGrath et al. “Bifröst: Visualizing and checking behavior of embedded systems across hardware and software”. In: *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. 2017, pp. 299–310.
- [138] David McNeill. *Hand and mind: What gestures reveal about thought*. University of Chicago press, 1992.
- [139] Ronald Metoyer et al. “Understanding the verbal language and structure of end-user descriptions of data visualizations”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2012, pp. 1659–1662.
- [140] Joseph E Michaelis et al. “Collaborative or Simply Uncaged? Understanding Human-Cobot Interactions in Automation”. In: *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 2020, pp. 1–12.
- [141] Harlan D Mills. “Software engineering education”. In: *Proceedings of the IEEE 68.9* (1980), pp. 1158–1162.
- [142] Audris Mockus and James D Herbsleb. “Expertise browser: a quantitative approach to identifying expertise”. In: *Proceedings of the 24th international conference on software engineering*. ACM. 2002, pp. 503–512.
- [143] “MutationObserver”. In: Accessed: March, 2020. URL: <https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver>.
- [144] Brad Myers et al. “How designers design and program interactive behaviors”. In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE. 2008, pp. 177–184.
- [145] Mitchell J Nathan, Kenneth R Koedinger, Martha W Alibali, et al. “Expert blind spot: When content knowledge eclipses pedagogical content knowledge”. In:

- [146] Jasper O’Leary et al. “Charrette: Supporting In-Person Discussions around Iterations in User Interface Design”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–11.
- [147] Ohan Oda et al. “Virtual replicas for remote assistance in virtual and augmented reality”. In: *UIST*. 2015.
- [148] Gary M. Olson and Judith S. Olson. “Distance Matters”. In: *Human-computer interaction* 15.2 (2000), pp. 139–178.
- [149] Stephen Oney, Brad Myers, and John Zimmerman. “Visions for Euclase: Ideas for Supporting Creativity through Better Prototyping of Behaviors”. In: *ACM CHI 2009 Workshop on Computational Creativity Support*. 2009.
- [150] Steve Oney, Christopher Brooks, and Paul Resnick. “Creating guided code explanations with chat. codes”. In: *Proceedings of the ACM on Human-Computer Interaction* 2.CSCW (2018), pp. 1–20.
- [151] Steve Oney et al. “Implementing Multi-Touch Gestures with Touch Groups and Cross Events”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12.
- [152] Sergio Orts-Escolano et al. “Holoportation: Virtual 3d teleportation in real-time”. In: *UIST*. 2016.
- [153] Jiazhi Ou et al. “Gestural communication over video stream: supporting multimodal interaction for remote collaborative physical tasks”. In: *Proceedings of the 5th international conference on Multimodal interfaces*. ACM. 2003.
- [154] Stack Overflow. *Stack Overflow*. Accessed: September, 2015. 2015. URL: <https://stackoverflow.com/>.
- [155] John F Pane, Brad A Myers, et al. “Studying the language and structure in non-programmers’ solutions to programming problems”. In: *International Journal of Human-Computer Studies* 54.2 (2001), pp. 237–264.
- [156] Jungkook Park, Yeong Hoon Park, and Alice Oh. “Non-Linear Editing of Text-Based Screencasts”. In: *The 31st Annual ACM Symposium on User Interface Software and Technology*. ACM. 2018, pp. 403–410.

- [157] Sun Young Park, Brad Myers, and Amy J Ko. “Designers’ natural descriptions of interactive behaviors”. In: *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE. 2008, pp. 185–188.
- [158] Amy Pavel et al. “VidCrit: video-based asynchronous video review”. In: *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 2016, pp. 517–528.
- [159] Tomislav Pejisa et al. “Room2room: Enabling life-size telepresence in a projected augmented reality environment”. In: *Proceedings of the 19th ACM conference on computer-supported cooperative work & social computing*. 2016, pp. 1716–1725.
- [160] Zhenhui Peng et al. “Design and evaluation of service robot’s proactivity in decision-making support process”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–13.
- [161] Maria Soledad Pera and Yiu-Kai Ng. “A community question-answering refinement system”. In: *Proceedings of the 22nd ACM conference on Hypertext and hypermedia*. ACM. 2011, pp. 251–260.
- [162] Anita Pincas. “Successful online course design: Virtual frameworks for discourse construction”. In: *Educational Technology & Society* 1.1 (1998), p. 15.
- [163] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. “Seahawk: Stack overflow in the ide”. In: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. 2013, pp. 1295–1298.
- [164] Rafael La Porta et al. *Trust in large organizations*. Tech. rep. National Bureau of Economic Research, 1996.
- [165] Zhengrui Qin et al. “Mobiplay: A remote execution based record-and-replay tool for mobile applications”. In: *Proceedings of the 38th International Conference on Software Engineering*. 2016, pp. 571–582.
- [166] Daniel Rakita, Bilge Mutlu, and Michael Gleicher. “An autonomous dynamic camera method for effective remote teleoperation”. In: *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*. 2018, pp. 325–333.

- [167] William T. Reeves. “Inbetweening for Computer Animation Utilizing Moving Point Constraints”. In: *Proceedings of the 8th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’81. Dallas, Texas, USA: Association for Computing Machinery, 1981, pp. 263–269. ISBN: 0897910451. DOI: [10.1145/800224.806814](https://doi.org/10.1145/800224.806814). URL: <https://doi.org/10.1145/800224.806814>.
- [168] Daniela Retelny et al. “Expert crowdsourcing with flash teams”. In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM. 2014, pp. 75–85.
- [169] Fatemeh Riahi et al. “Finding expert users in community question answering”. In: *Proceedings of the 21st international conference companion on World Wide Web*. ACM. 2012, pp. 791–798.
- [170] Martin P Robillard, Robert J Walker, and Thomas Zimmermann. “Recommendation systems for software engineering”. In: *Software, IEEE 27.4* (2010), pp. 80–86.
- [171] Universal Robots. Accessed: Sep., 2020. 2020. URL: <https://www.universal-robots.com/>.
- [172] Hugo Romat et al. “SpaceInk: Making Space for In-Context Annotations”. In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 2019, pp. 871–882.
- [173] Julia Schenk, Lutz Prechelt, and Stephan Salinger. “Distributed-Pair Programming can work well and is not just Distributed Pair-Programming”. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 74–83.
- [174] Scrimba. In: <https://scrimba.com/>. 2019.
- [175] Rita Shewbridge, Amy Hurst, and Shaun K Kane. “Everyday making: identifying future uses for 3D printing in the home”. In: *Proceedings of the 2014 conference on Designing interactive systems*. ACM. 2014, pp. 815–824.
- [176] Jonathan Sillito, Gail C Murphy, and Kris De Volder. “Asking and answering questions during a programming change task”. In: *Software Engineering, IEEE Transactions on 34.4* (2008), pp. 434–451.
- [177] Devrim Yasar Sinan Yasar. *Koding*. Accessed: September, 2015. 2012. URL: <https://koding.com>.

- [178] Snap. “Discover”. In: <https://support.snapchat.com/en-US/a/discover>. 2018.
- [179] Rajinder S Sodhi et al. “BeThere: 3D mobile collaboration with spatial input”. In: *CHI*. 2013.
- [180] Igor de Souza Almeida et al. “AR-based video-mediated communication: a social presence enhancing experience”. In: *Virtual and Augmented Reality (SVR), 2012 14th Symposium on*. IEEE. 2012, pp. 125–130.
- [181] Aaron Stafford, Wayne Piekarski, and Bruce Thomas. “Implementation of god-like interaction techniques for supporting collaboration between outdoor AR and indoor tabletop users”. In: *ISMAR*. 2006.
- [182] Igor Steinmacher, Marco Aurélio Graciotto Silva, and Marco Aurélio Gerosa. “Barriers faced by newcomers to open source projects: a systematic review”. In: *Open Source Software: Mobile Open Source Technologies*. Springer, 2014, pp. 153–163.
- [183] Jonathan Steuer. “Defining virtual reality: Dimensions determining telepresence”. In: *Journal of communication* 42.4 (1992), pp. 73–93.
- [184] Mengü Sukan et al. “Providing Assistance for Orienting 3D Objects Using Monocular Eyewear”. In: *SUI*. 2016.
- [185] Amanda Swearngin et al. “Rewire: Interface design assistance from examples”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 2018, pp. 1–12.
- [186] Matthew Tait and Mark Billingham. “The effect of view independence in a collaborative AR system”. In: *CSCW* (2015).
- [187] Anthony Tang, Michael Boyle, and Saul Greenberg. “Display and presence disparity in Mixed Presence Groupware”. In: *Proceedings of the fifth conference on Australasian user interface*. 2004.
- [188] Kesler Tanner, Naomi Johnson, and James A Landay. “Poirot: A Web Inspector for Designers”. In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 2019, pp. 1–12.

- [189] Franco Tecchia, Leila Alem, and Weidong Huang. “3D helping hands: a gesture based MR system for remote collaboration”. In: *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM. 2012, pp. 323–328.
- [190] Rannie Teodoro et al. “The motivations and experiences of the on-demand mobile workforce”. In: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 2014, pp. 236–247.
- [191] Balasaravanan Thoravi Kumaravel et al. “Loki: Facilitating Remote Instruction of Physical Tasks Using Bi-Directional Mixed-Reality Telepresence”. In: *UIST*. 2019.
- [192] Edward R Tufte. *The visual display of quantitative information*. Vol. 2. 1983.
- [193] Ufactory. Accessed: Sep., 2020. 2020. URL: <https://www.ufactory.cc/#/en/>.
- [194] *Upwork Inc. (formerly oDesk)*, <https://www.upwork.com>. Accessed: April, 2016. 2015. URL: <https://www.upwork.com>.
- [195] DWF Van Krevelen and Ronald Poelman. “A survey of augmented reality technologies, applications and limitations”. In: *International journal of virtual reality* 9.2 (2010), p. 1.
- [196] April Yi Wang et al. “Callisto: Capturing the “Why” by Connecting Conversations with Computational Narratives”. In: *CHI '20 (2020)*.
- [197] Yang Wang et al. “Flying eyes and hidden controllers: A qualitative study of people’s privacy perceptions of civilian drones in the US”. In: *Proceedings on Privacy Enhancing Technologies* 2016.3 (2016).
- [198] Patricia G Wojahn, Christine M Neuwirth, and Barbara Bullock. “Effects of interfaces for annotation on communication in a collaborative task”. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1998, pp. 456–463.
- [199] Erroll Wood et al. “Shadowhands: High-fidelity remote hand gesture visualization using a hand tracker”. In: *ISS*. 2016.
- [200] Yaxing Yao et al. “Privacy Mechanisms for Drones: Perceptions of Drone Controllers and Bystanders”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 6777–6788. ISBN: 9781450346559.

- [201] Dongwook Yoon et al. “RichReview: blending ink, speech, and gesture to support collaborative document review”. In: *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 2014, pp. 481–490.
- [202] Dongwook Yoon et al. “RichReview++ Deployment of a Collaborative Multi-modal Annotation System for Instructor Feedback and Peer Discussion”. In: *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*. 2016, pp. 195–205.
- [203] Jun Zhang, Mark S Ackerman, and Lada Adamic. “Expertise networks in online communities: structure and algorithms”. In: *Proceedings of the 16th international conference on World Wide Web*. 2007, pp. 221–230.
- [204] Jun Zhang et al. “QuME: a mechanism to support expertise finding in online help-seeking communities”. In: *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 2007, pp. 111–114.
- [205] Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. “Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR”. In: *ISMAR*. 2008.
- [206] Joyce Zhu et al. “Toward a domain-specific visual discussion forum for learning computer programming: An empirical study of a popular MOOC forum”. In: *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2015, pp. 101–109.