

Improving User Involvement Through Live Collaborative Creation

by

Sang Won Lee

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2018

Doctoral Committee:

Professor Walter S. Lasecki, Chair
Professor Georg Essl, Co-Chair
Professor Mark Ackerman
Professor Steven Oney

Sang Won Lee

snaglee@umich.edu

ORCID iD:0000-0002-1026-315X

© Sang Won Lee 2018

A C K N O W L E D G M E N T S

First of all, I would like to thank my advisors, Prof. Walter Lasecki and Prof. Georg Essl. Their excellence and passion in research have helped me develop the research vision that motivates me, learn how I can contribute to the outer world with it, and finally make me determined to continue research. I am also very grateful to Prof. Mark Ackerman and Prof. Steve Oney for serving my dissertation committee. Their constructive and critical feedback throughout the process strengthen the dissertation, continuously helping prepare the better version of myself. I was lucky enough to have many mentors and want to extend my thanks and gratitude to them, including Prof. Jason Freeman, Prof. Michael Gurevich, Prof. Sile O’Modhrain, Dr. Hongchan Choi, and Prof. Alan Blackwell. I am grateful to my excellent collaborators that I had a chance to work with during my doctoral study — Antonio Deusany de Carvalho Junior, Yan Chen, Stephanie O’Keefe, Rebecca Krosnick, Mari Martinez, Jungho Bang, Ben Taylor, Jeff Scott, and D. Andrew Stewart. I would like to thank my friends and colleagues who have inspired, challenged, and supported me throughout the graduate study. My parents, Haisook Kong and Myun Woo Lee, are the unsung heroes in this lengthy battle. I would have not even dreamed of what I am today without them. Last but not least, I would like to thank my two lovely housemates — my wife, Sol Ie Lim, and my son, Jason Lee — for being my last resort. I cannot wait to take our exciting journey together.

TABLE OF CONTENTS

Acknowledgments	ii
List of Figures	vii
List of Tables	xii
Abstract	xiii
Chapter	
1 Introduction	1
1.1 Live Collaborative Creation as User Involvement	1
1.2 Definition of <i>Live</i>	3
1.3 Definition of <i>Liveness</i>	6
1.4 Live Collaborative Creation with End Users	8
1.5 Problem Statement and Thesis Statement	10
1.6 Identifying and Addressing Challenges in Live Collaboration	12
1.6.1 Qualitative User Study on Real-time Collaboration between an Expert and Novices in Crowdsourcing	13
1.6.2 Environments for Live Collaborative Programming	14
1.7 Lowering the Barriers to Live Collaborative Creation	15
1.7.1 <i>SketchExpress</i> : Remixing Animations for Crowd-powered Pro- totyping of Interactive interfaces	16
1.7.2 <i>Crowd in C</i> : Audience Participation in a Musical Performance Using Smartphones and Cloud Computing	17
1.8 Liveness for Asynchronous Communication and Collaboration	17
1.8.1 <i>CodeOn</i> : On-demand Programming Assistance	18
1.8.2 <i>Live Writing</i> : Accessing Real-time History via Record-and-replay in Writing	20
1.9 Contributions and Outline	20
2 Related Work	23
2.1 Liveness in Programming Environments	23
2.2 Preserving Liveness in Groupware	24
2.2.1 Timeline and Replay in Programming and Writing	24
2.2.2 Asynchronous Change Awareness	25
2.2.3 Networked Live Coding for Collaboration	26
2.3 Collaboration in Crowdsourcing	28

2.3.1	Real-time Collaboration in Crowdsourcing	28
2.3.2	Facilitating Communication in Crowdsourcing	29
2.3.3	Crowdsourced Music Performances Using Smartphones	30
2.3.4	Observational Studies in Real-time Collaboration	31
3	A User Study on Real-Time Collaboration in a Crowd-powered Sketching Tool	32
3.1	Introduction	32
3.2	Apparition — System Description	34
3.3	User Study	37
3.3.1	Study Procedure	38
3.3.2	Participants	39
3.3.3	Data Analysis	40
3.3.4	Limitations	41
3.4	Result	42
3.4.1	Real-time Collaboration with Workers in an Interactive Crowd-sourcing System	43
3.4.2	Expertise, Language, and Jargon	45
3.4.3	Speech, Text, Asymmetric Communication	48
3.4.4	Challenges in Co-working on a Shared Artifact	51
3.4.5	Varying Pace of Making Requests	53
3.5	Design Recommendations	55
3.5.1	Turn Live Speech into a Structured Task List	55
3.5.2	Address Asymmetry in Communication and Expertise	56
3.5.3	Protect Requester Artifacts in Shared Space	56
3.5.4	Augment Workspace Awareness of Presence and Implicit Interactions	57
3.5.5	Train Workers to Collaborate, Not Just Use	57
3.6	Conclusion	58
4	Tools for Live Collaborative Programming	59
4.1	Motivation: Live Coding the Mobile Music Instrument	59
4.2	a Programming Environment for Collaborative, Live Coding	61
4.2.1	Centralized State Space	61
4.2.2	Individual and Shared Namespace	62
4.2.3	Live Variable View	64
4.2.4	Distributed Code Execution through Chat Communication	66
4.3	Exploring Real-time Coordination Tools for Ad Hoc Programming Teams	67
4.3.1	Ad Hoc Crowd Programming Teams	67
4.3.2	Coordination Costs in Ad Hoc Teams	68
4.3.3	Identifying the needs of self-coordination	69
4.3.4	A Shared Code Editor for Ad-Hoc Teams	72
4.4	Conclusion	74
5	SketchExpress: Crowdsourcing Interactive Behaviors in GUI Sketch Prototype	75
5.1	Introduction	75

5.2	Challenges in Prototyping Interactive Behaviors	76
5.2.1	Designing Interactive Behaviors in Sketching Tools	77
5.2.2	Programming by [Remixing] Demonstration	79
5.2.3	Summary of Tools for Prototyping Interactive Behaviors	79
5.3	SketchExpress: System Description	80
5.3.1	Platform	82
5.3.2	Recording Demonstration	82
5.3.3	Recording and Replaying By Delta	83
5.3.4	Remixing Animations	84
5.3.5	Animation as a First Class Object	87
5.4	Laboratory User Evaluation	87
5.4.1	User Study - UI Tasks	87
5.4.2	Participants	88
5.4.3	Experimental Design	89
5.4.4	Performance Measures	90
5.5	Result: a Sketch That Behaves	91
5.5.1	Improving Quality	91
5.5.2	Improving Long-Term Latency	91
5.5.3	Task Engagement	93
5.6	Conclusion	94
6	Crowd in C: Audience Participation in Music Performance	95
6.1	Introduction	95
6.2	Motivation: Crowd Playing and Mingling in C	96
6.3	Collective Creation of Music	97
6.3.1	Loop-based Instrument	97
6.3.2	Social Interaction with Online Dating Metaphor	98
6.4	Crowd Musicking in Cloud	101
6.4.1	Mobile Ad Hoc Network using Cloud Service	101
6.4.2	Orchestrating the Crowd through Live Coding	103
6.5	Validation - Crowd in C in Practice	104
6.6	Future Work: Gauging and Understanding Engagement through Performance Analysis	106
6.7	Conclusion	107
7	Codeon: On-Demand Software Development Assistance	108
7.1	Motivation	108
7.2	Related Work	109
7.2.1	Help Seeking in Software Development	110
7.2.2	IDE-Integrated Help Finding Tools	112
7.2.3	Human Expert Computation	113
7.3	Codeon : Implementation	113
7.4	Evaluation	117
7.4.1	Experimental Setup	117
7.4.2	Results	118

7.4.3	<i>Interruptions and Parallelization</i>	120
7.4.4	Post Interview and Developer Feedback	122
7.5	Conclusion	124
8	Live Writing: Preserving Liveness in Asynchronous Written Communication	125
8.1	Introduction	125
8.2	Motivation	126
8.2.1	Keystroke Logging and Playback	126
8.2.2	Context Recovery and Task Resumption in Collaborative Writing	127
8.2.3	Access to Real-time History of a Document	127
8.2.4	Writing as a Real-time Performance Art	128
8.3	Live Writing - Capturing Liveness of Writing	129
8.4	Future Work	131
8.4.1	Understanding Effects of Liveness	131
8.4.2	Notation Mechanism for Live Coding and Writing	133
8.4.3	Intelligent Writing Systems	134
9	Conclusion	136
	Bibliography	139

LIST OF FIGURES

1.1	Examples of <i>non-live</i> settings: Typically, there is a clear separation between the process of creating an artifact and consuming it. In other words, the process of creation is not live, which means that the creation process is not immediately perceptible to users.	5
1.2	Examples of live creation: In teppan-yaki restaurants, customers can see the process of creating an artifact (top left). At concerts, audience members can see musicians performing in front of them in real time (bottom left). In some restaurants, customers can actively participate in the cooking process by choosing ingredients (top right). Some music is composed to incorporate audience participation, such as Queen’s “We Will Rock You”, in which audiences are guided to stomp and to clap to the rhythm (bottom right).	9
1.3	An artifact is typically consumed long after the time it was created. My research explores live collaborative creation in which users can be involved in the creative process. There are three subgoals that I present: 1) to identify and address the challenges in computationally supporting live collaborative creation, 2) to develop tools and methods that can reduce the barriers of entry to live collaborative creation for non-experts and broaden the applications of live collaborative creation, and 3) to bring liveness into asynchronous communication when we cannot collaborate live.	12
1.4	Real-time collaboration between a requester and crowdworkers for a GUI sketch prototype: 1) A requester verbally describes and draws a rough sketch on a shared drawing tool in real time. 2) A set of crowdworkers listens to the verbal description with the rough sketch. 3) The crowdworkers enhance the rough sketch into a prototype with higher fidelity, and communicate with the requester through the chat interface.	14
1.5	A programming environment for collaborative, live coding. The live variable view in the top left corner makes the program state visible to all connected programmers in real time. Note that each programmer has a unique namespace to avoid spontaneous naming conflicts between variables and functions.	15
1.6	<i>SketchExpress</i> : Non-expert crowd workers can create interactive behaviors in a few minutes using the demonstrate-remix-replay method. The final sketch will have a set of animations that demonstrate multiple behaviors.	16

1.7	A web-based musical instrument for the audience participation music piece <i>Crowd in C[loud]</i> . The short composition (left) serves as a profile on a social network, and users can browse other people’s profiles (middle) and play together (right). The social interaction among audience members helps sustain their interest for the duration of the performance.	18
1.8	Codeon: the code context that is associated with a programming question is visible to an expert crowdworker.	19
1.9	Live Writing: A web-based text editor that can record and replay writing activity in real time. This type of screencast preserves textual information that users can use.	21
1.10	Live Writing: Gloomy Streets. Writing a poem on stage live in front of the audience becomes an audiovisual performance. The piece uses temporal typography to enhance communication with the audience.	21
3.1	The requester interface of SketchExpress supplies a variety of tools that facilitate sketching and communication with crowd workers: (a) shared canvas, (b) pencil tool, (c) selection tool, (d) chat box, (e) mute/unmute button, (f) circles to visualize requester click events.	35
3.2	(Given Sketch) The sketch of a to-do list application given to participants (requesters). (T1) A sketch hand-drawn in SketchExpress in Task 1 without crowd workers. (T2) Progression of a sketch in SketchExpress in Task 2 as T2-a) it is first hand-drawn by the requester (same drawer as in T1), T2-b) the requester continues hand-drawing and crowd workers begin replacing hand-drawn pieces with higher-fidelity shapes and icons, and T2-c) the crowd workers replace all hand-drawn pieces.	37
3.3	On average, the non-designers described the request in more descriptive ways and referred to elements more frequently when working with workers	46
3.4	<i>Hamburger menu gone wrong</i> : P1 used the jargon “hamburger menu” and a crowd worker spelled it out, as they were not familiar with the term.	47
3.5	9 min-long history of chat during P7’s session. 1) Browsing the chat history does not help recover the context. 2) The worker asks for permission to delete the requester’s drawings (line 1). 3) The worker cannot recall the request (line 3 and 8).	51
3.6	a) <i>Crowded canvas</i> : The workers place their higher-fidelity square checkboxes and task label text directly on top of the corresponding elements hand-drawn by the requester, instead of deleting the hand-drawn elements first. b) <i>Split canvas</i> : The crowd workers choose to preserve the requester’s drawing and create their prototype next to it.	52

3.7	(1),(2): the number of words spoken so far(y-axis) over time (x-axis) for P2 (designer) and P9 (non-designer). The orange (longer) line indicates T2 and the blue (shorter) one indicates T1. For T2, P9 speaks in constant pace throughout the session, whereas P2 finished the initial requests quickly and stay silent for a while. (3),(4): response delay over time. (x-axis): the timestamp at which an initial request was made per element, (y-axis): the time (delay) it took for crowd workers to respond to the request. Delay for P9 showed a linear trends, which indicates the requests got quickly backlogged.	54
4.1	In this performance practice, two programmers develop an application in real time by sending code text to an application that is running on a tablet over a wireless network. The application, which is a mobile music instrument, is used by a musician at the same time. While programmers can write and send code to modify the program state, the program state that changes by their own code, collaborators' code and user's interaction is not clear. This obscurity is an obstacle to live collaboration. I suggest a tool that makes the program state visible to the programmers, depicted as orange arrows in the diagram.	60
4.2	Centralized architecture of collaborative, live coding in urMus	62
4.3	a) There is only one state space leaving open the risk you inadvertently modify someone else's state space. See there are two count variables, the later produced of which will overwrite the earlier one. b) Having a separate namespace per each live coder will prevent this collision but then how will they collaborate without shared state? c) There is a shared namespace that all live coders have access to. A live coder can share either a variable or function to be shared with other live coders. For example, Bob can choose to share a function "set-Frequency" so that Alice can execute that function.	63
4.4	Live Variable View. Individual programmers' live program state are displayed, and they can share selected objects with collaborators.	65
4.5	a shared editor with subtask view. 1) log in 2) a shared editor with region boundaries 3) subtask-view 4) input data form 5) console output 6) chat interface 7) run button	73
5.1	SketchExpress allows crowd workers to prototype interactive behaviors. A requester describes aloud how a user interface should behave and crowd workers quickly create complex interactive behaviors. The interface contains the following features for crowd workers to easily create interactive behaviors (animations) as follows: (1) A synchronized canvas that supports simultaneous interactions between a requester and workers. (2) the ability to select and replay multiple animations at once; (3) reset functionality that places elements in the animation back to their initial state (position, color, etc.); (4) recording button to record a worker's demonstrations; (5) a chat box for helpers to ask clarification questions if needed. (6) labels that show the current state of the animation [replaying/remixing] to prevent multiple workers from concurrently working on the same animation.	81

5.2	(1) Initial State: arrow pointed vertically upwards. (2) After first replay: arrow pointed 60 degrees clockwise. (3) After second replay: arrow pointed 120 degrees clockwise. Reset will restore state (1).	83
5.3	SketchExpress facilitates the process of creating complex animations involving multiple transformations on one element. In this example, a worker can create separate <i>rotate</i> and <i>translate</i> operations, and then replay them together to create the rolling stone animation.	84
5.4	Remixing helps workers make more expressive animations. The interface consists of: (1) Operation list: provides workers with a discrete view of an animation as a series of operations. (2) Operation duration: if clicked, a container of remix functions is expanded. (3) Replay options: you can choose for each element if it will be skipped (not displayed), if it will appear instantly, or if it will appear in real-time. (4) Slider and input: modify the operation's speed and duration. (5) Trash icon: skip the operation or delay. (6) Check mark and highlight border: indicate the current operation in preview. (7) Element replacement: reuses the animation for one element as the animation for another one.	86
5.5	Though there is not a significant difference in recall between (C1) and (C2), recall is significantly higher in (C3) compared to (C1) and (C2) (both $p < .001$). There are significant increases in precision from C1 to C2 ($p < .001$) and from C2 to C3 ($p < .001$).	92
5.6	Latency of the 1st demo(blue) and the 2nd demo(yellow); The time it takes to demo-remix-replay an animation is longer than the other two conditions, but once an animation is created (the 2nd demo), a worker can respond quickly by replaying the animation. This is because the amount of time needed to respond to the demonstration request is the time it takes to restore the initial state needed to reproduce the requester behavior (the portion of the green bar in the yellow one).	93
6.1	Screen-shots of Audience Interface. Playing Ascending Tones (left) and Drawing a Star (right).	99
6.2	Screenshots of Audience Interface in Five States. From left to right: NAME, EDIT, WAIT, CHECK, MINGLE states	100
6.3	a Screen-shot of Performer Interface	101
7.1	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).	114
7.2	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)	115

7.3	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).	116
7.4	Overall result: The # of completed tasks is significantly more in Codeon condition(avg./s.d.).	118
8.1	Screenshot of Live Writing Website.	130
8.2	Changes are highlighted in Microsoft Words for change awareness in collaborative writing.	133

LIST OF TABLES

4.1	Six experiments (E1-E6) are ran in different conditions. For condition 2(C2), participants used a shared editor and for condition 3(C3), participants used a version control system (git). W indicates a crowd worker and S indicates a university student.	70
7.1	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.	119
7.2	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.).	120
7.3	# of interruptions, alerts, and average of individual ratio of interruption/alert(Avg. / S.D.).	120
7.4	Time spent and the number of parallelization behavior in two conditions (Avg. / S.D.).	122

ABSTRACT

Creating an artifact — such as writing a book, developing software, or performing a piece of music — is often limited to those with domain-specific experience or training. As a consequence, effectively involving non-expert end users in such creative processes is challenging. This work explores how computational systems can facilitate collaboration, communication, and participation in the context of involving users in the process of creating artifacts, while mitigating the challenges inherent to such processes. In particular, the interactive systems presented in this work support *live collaborative creation*, in which artifact users collaboratively participate in the artifact creation process with creators in real time. In the systems that I have created, I explored liveness, the extent to which the process of creating artifacts and the state of the artifacts are immediately and continuously perceptible, for applications such as programming, writing, music performance, and UI design. Liveness helps preserve natural expressivity, supports real-time communication, and facilitates participation in the creative process. Live collaboration is beneficial for users and creators alike: making the process of creation visible encourages users to engage in the process and better understand the final artifact. Additionally, creators can receive immediate feedback in a continuous, closed loop with users. Through these interactive systems, non-expert participants help create such artifacts as GUI prototypes, software, and musical performances. This dissertation explores three topics: (1) the challenges inherent to collaborative creation in live settings, and computational tools that address them; (2) methods for reducing the barriers of entry to live collaboration; and (3) approaches to preserving liveness in the creative process, affording creators more expressivity in making artifacts and affording users access to information traditionally only available in real-time processes. In this work, I showed that enabling collaborative, expressive, and live interactions in computational systems allow the broader population to take part in various creative practices.

CHAPTER 1

Introduction

1.1 Live Collaborative Creation as User Involvement

People use and consume various artifacts in their daily lives, such as by reading a book, watching TV shows, using software, and even going out for dinner. For creators—e.g., designers, developers, artists—the process of creating artifacts is a complicated one in which creators search for ways to ensure that the artifact fulfills users’ needs and eventually gives them an engaging user experience. However, users needs are often implicit and challenging to identify.

One way to better understand what users want is to involve them in the design and development process coming from a long tradition of user-centered design methods to involve users [1]. Using these methods, researchers, designers and developers can involve users at various stages of the design process. In addition, the level of involvement can be varied in terms of roles and methods, from being observed to directly participating in the design process. For example, usability testing is a widely used method to collect feedback from users to learn how to improve an artifact by letting them use and explore it in a controlled setting [2, 3]. In participatory design [4], users are involved even before creating the artifact and are invited to participate in the design process. In another method of contextual design, researchers and developers visit the context that users live and work in via ethnography and contextual inquiry [5, 6, 7].

The eventual goals of user involvement are for creators to improve their artifacts and to provide better user experiences to users. Damodaran [8] stated that effective user involvement yields benefits, including having more accurate user requirements, avoiding unnecessary features, reducing friction in accepting the system, and enhancing understanding of the artifact.

However, the effects of user involvement can be lagged by the lengthy process. Typical user involvement methods are a priori methods, carried out *before* the artifact is used.

Often, the user involvement process is too time-consuming and labor-intensive [9]. The turnaround time from the user involvement activity to the time when the artifact can actually be improved with the results can take a long time. It is because of the iterative nature that the result of user involvement needs to be fed back to the design process and it needs to be repeated multiple times to again improve the revised artifact [10]. Therefore, the participants that are involved in the creation process do not, at least not immediately, benefit from their involvement. Minimizing the delay between involving users and delivering the effects of it is thus desirable.

One of the common approaches is to reduce the time that it takes to conduct various user involvement methods. Hughes et al. developed a focused ‘quick and dirty’ ethnography to gain a rapid understanding of the field [11]. Milen introduced a similar method of “rapid ethnography” that can be carried out in a limited amount of time in the field [12]. Holtzblatt et al. suggest Rapid Contextual Design (RCD), adapting contextual design to tactical projects with tight time constraints and resources [13]. Rapid and iterative testing and evaluation (RITE) is a business-centered usability testing method that suggests continually improving an artifact during testing and evaluation [14]. In all of these methods, shortening the time required to involve users can help to reduce the delay. However, all these approaches are far from accomplishing immediacy, taking time on the order of months.

One of the ways we can achieve the immediate effects of user involvement is to give participants an engaging experience throughout the involvement process. However, existing methods of user involvement have been focused more on observing and understanding participants’ behavior rather than facilitating their engagement. Inherently, user involvement settings involve an unnatural environment containing observers (or their equivalent, such as video camcorders). For example, participants in usability testing are asked to perform a task they are not familiar with, and to verbally justify their behaviors while being observed. Industry usability specialists suggest including more direct and natural communication with users as opposed to in-lab usability testing, because the artificial laboratory setting causes tension and nervousness on participants’ end [15]. Likewise, it takes long time for an ethnographic researcher to sufficiently be integrated into the community for participants to trust them [16]. This reflects how the ethnography method can be seen as an awkward intervention of a researcher for participants. This challenge has been discussed mostly for the purpose of ensuring the *ecological validity* of the outcomes of a experiment that involves human subjects in a laboratory setting [17]. Nevertheless, it has not been actively discussed in the context of user involvement potentially resulting in an unengaging and even intimidating experience for those involved. Audience involvement in public events (music performances, can be a good example of which the goal is to engage those who are involved

in the process, attempted in various fields, from music to education [18, 19, 20]

In this dissertation, I have explored computational systems that support *live collaborative creation* as a new way of user involvement to address the issues of latency and disengagement. In the proposed method, creators involve users in the creation process at the time when artifacts are created by offering computational systems that mediate real-time collaboration between users and creators. The effects of user involvement are immediate and direct: the goal of this live, collaborative creation is to give those who are involved an engaging experience through participation. Live creative collaboration is composed of two components: 1) *live creation*: revealing the creation process in real-time; and 2) *real-time collaboration*: end users being involved in the creation process by collaborating directly with creators.

1.2 Definition of *Live*

Throughout this dissertation, the term *live* will be used frequently. Recently, interest in computational systems and digital media that include live interaction have been growing in the field of human-computer interaction [21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34]. However, the term *live* is not defined explicitly in the context of computational systems and media. In the Oxford Dictionary, the adjective *live*, in this context, means “relating to a musical performance given in concert, not on a recording” or “transmitted at the time of occurrence, not from a recording”. In the Merriam-Webster Dictionary (online), *live* is defined as “of or involving a presentation (such as a play or concert) in which both the performers and an audience are physically present” or “broadcast directly at the time of production”. Two common components of the above definitions are the concurrence of action and perception, and the notion of an artifact that is produced, performed, or transmitted, such as a musical performance, TV show, or sporting event. The concurrence of action and perception implies live settings include two different groups: creators — those who are in action (performers, actors, broadcasters) — and users (consumers more broadly) — those who perceive the action (an audience, viewers, listeners). In summary, there exist four components that constitutes being live: 1) an artifact, 2) those who create/deliver the artifact, 3) those who perceive it, and 4) the concurrence between 2 and 3.

In this dissertation, we define the term *live* as follows:

relating to the process of creating or delivering an artifact being perceptible in real time to spectators.

Therefore, the artifact that is being created or, oftentimes, the process (e.g. a musical

performance, which is itself an artifact) should be also perceptible, typically visible and audible. First, the term “perceptible” is used in its broadest sense to include any capability of being perceived through any human sensory system (e.g., visible, audible, tangible, smellable, tastable). For example, a live concert can be heard over the radio. Frequently, the term live relates to the perception of multisensory information, typically audiovisual.

Second, being perceptible in “real time” means that the process is perceptible immediately (exactly at the time of occurrence) or with minimal delay. Interestingly, the notion of liveness has developed with the advancement of media technologies through which one can transmit audiovisual information to a remote location (radio, “tele”phone, “tele”vision, and recording). Therefore, the term “real time” typically relates to the latency involved in technologies that enable the visibility (or perceptibility more precisely) of the process and artifacts to remote spectators. Otherwise, any live process for which spectators and performers are co-located is visible in real-time physically.

Typically, it cannot be immediate due to the latency involved in telecommunication and data transmission. However, the amount of acceptable latency may vary from one context to another. Live broadcasting on a television is less sensitive to the length of the latency, as television broadcasts are a form of one-way communication. Even tens of seconds delay may not be recognizable, and do not interfere with the user experience as long as the media delivery is continuous and the latency is consistent. Sometimes, having a long latency is even desirable; in typical live TV shows, a broadcast delay is intentionally added to allow undesirable content, such as profanity or nudity, to be censored [35]. The spectators can perceive a live process in near real-time, or with some delay if necessary. However, in the case of an interactive system (two-way communication) beyond media delivery, latency can negatively impact the user experience. For example, a teleconferencing system (such as Skype) allows creators to present the process of creation live to users, but a delay of more than a few seconds would make two-way communication difficult [36].

Typically creation processes are not live — in fact, they are often *hidden* to users entirely. The creation of an artifact takes place asynchronously with respect to its consumption, and the process of creation is separated from the users. For example, readers (users) read a book (an artifact) only after the author (its creator) has finished writing it. The same is typically true of food that people eat at a restaurant, a painting exhibited at a museum, or software written by developers. On the contrary, in live settings, the process of creating artifacts is revealed to users to an extent that is typically understandable and presentable to spectators. For example, at a live music concert, music is the *artifact*, musicians are the *creators*, and audience members are the *users*. While many aspects of a music performance are still asynchronously done or hidden — such as practices, rehearsals, back-stage efforts —

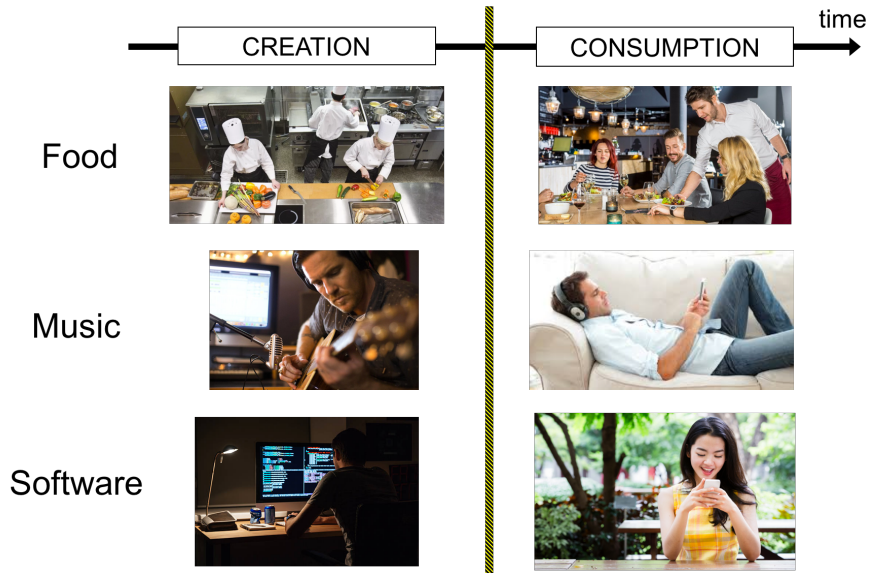


Figure 1.1: Examples of *non-live* settings: Typically, there is a clear separation between the process of creating an artifact and consuming it. In other words, the process of creation is not live, which means that the creation process is not immediately perceptible to users.

the audience get to see the actual performance and perceive it as a live creation of an artifact. Such asynchronous efforts are more necessary for especially making the process live.

Many creators choose to reveal the process of creation to users so that users can understand how an artifact is made and appreciate the effort that goes into creating an artifact. In general, people put more value on an artifact that is created live. For instance, people are willing to pay more for live music performances, even though listeners are less and less willing to pay for recorded music [37]. In the case of cooking, in some restaurants, chefs cook in front of their customers in real time. This is true of teppan-yaki restaurants (sometimes referred to as “Japanese steakhouses” in the US), which serve a style of Japanese cuisine cooked on an iron hot plate called a *teppan* [38]. See Figure 1.2 for more examples of live settings. The visible process of creation adds value to the artifact because users can understand how the artifact is created [39].

Lastly, live creation encourages creators to reinforce the performative aspect of the creation process. Creators would want to augment their actions (i.e., narration) and to add even unnecessary steps or modifications (i.e., exaggerated movements) for effective communication to make the process more transparent and engaging. The perceptibility on the artifact to spectators and the existence of them allows creators to express their thoughts and emotion on the artifact and to convey their creative practice. Increasing such demonstrative components of a live interaction gives creators opportunities to engage people through

understanding and theatrical elements. For users standpoint, as perceiving the process takes time, the artifact now is something that can be not only used but also experienced. Creators would “perform” the creation activity live like performing arts compared to the case of non-live creation with no audience.

1.3 Definition of *Liveness*

How, then, can we define *liveness*? According to the Oxford Dictionary liveness is “the quality or condition (of an event, performance, etc.) of being heard, watched, or broadcast at the time of occurrence.” In general, liveness has been valued, especially in the context of performing arts such as music, theatre, and dance. Auslander explored the value of liveness, particularly in the era of culture dominated by (typically non-live) mass media and media technology, such as television [39]. Liveness also has been expanded to broader areas digital arts and new media. Crisell also states that a broadcast conveying messages over distances without the time lapse is the basis of the broadcast’s liveness [40].

A non-live process can still incorporate liveness through the use of techniques that give the audience *the sense of being there* and the opportunity to witness what happens. A typical example of a non-live creative process that still has liveness is replay of recorded live events. Based on the definition of live used in the previous section, recording of a live music performance is NOT live due to the delay between creation and consumption. However, it was live at the time of recording, and the recording effectively shows the characteristics of live settings: a number of characteristics unique to live settings can be found, such as any risks involved, the impromptu nature of the performance, the sound of the audience cheering, and the unedited visuals, especially compared to non-live forms of them — studio-recorded music. Hook et al. describe liveness as “the properties of intimacy and immediacy experienced by both spectators and performers”, which comes from the spectators’ proximity to and even inclusion in a performance of some kind [21]. The authors assert that the uses of technology can bring a sense of presence and involvement in events, which can exist independently of time and space. For example, having multiple views of a live music concert on a television is impossible to replicate for an audience member at a concert hall. Therefore, liveness can be related to the qualities and properties that help users experience the live process of creating an artifact.

In this dissertation, I define *liveness* as follows:

<p><i>the extent to which the process of creating artifacts and the state of the artifacts are immediately and continuously perceptible</i></p>
--

This particular definition is selected in consideration of the context of interactive systems design and can be used to quantitatively assess liveness of an interactive system.

Liveness can be considered in terms of three values: *immediacy*, *continuity*, and *perceptibility*. The immediacy (or immediate perceptibility) can be simply measured by the average latency between the time of creation and the time that the process is perceptible to spectators. For example, live broadcasting of a music concert has more liveness than replaying a recording of the same concert, as the former becomes perceptible sooner after the time of creation. The continuity (or continuous perceptibility) is how continuously the state of the process is perceptible to spectators. If there is an artifact whose state is A, the state visible to spectators A' can be updated discretely or intermittently. The discrepancy between A and A' may arise not only from the latency, but also from the discontinuous visibility (or state synchronization) coming from the mediating technology. For example, a live TV sporting event with commercial breaks is not continuously visible, because viewers cannot see what happens during the breaks, unlike the audience attending the event in person. Text editors are another software example. Suppose one wants to write text live to remote viewers, for educational purposes. In this shared editor, the document is continuously shared — the state is synchronized with every keystroke — even though there can be some latency. However, in the case of an instant chat messenger, messages are shared only when a user presses the Return key (or presses a send button); until that point, the typed text is not visible to the other party. In this case, the artifact (conversation) is shared discontinuously. Therefore, Google Docs has more liveness than an instant chat messenger. The notion of continuity is particularly effective to understand the liveness when certain media or processes are not live and do not occur in real time — a recording of a live event, for instance.

As an extreme example, a film can be seen as a medium that has less liveness compared to a play in a theatre. Based on the definition used here, in terms of immediacy, the final artifact may be presented to viewers with months and years of latency. Another aspect that makes film have less liveness is the discontinuous ways in which the artifact is created and presented. The total time it takes to produce a film may span several years. However, only a tiny portion of the entire creative process is visible to the audience, as the typical running time of a film is from two to three hours. However, it takes one hour to perform an hour-long play, and it is visible for an hour to the audience. While a tremendous amount of pre-production process may be required (writing, scripting, rehearsal), the fact that the one-hour process is continuously visible to the audience in real time since the beginning constitutes liveness of a theatrical performance and brings with it the impromptu and risky nature of live performance. Similar kinds of liveness coming from continuity can exist in asynchronous filming. For example, a long take in a film can increase the liveness for a

particular scene.

Lastly, perceptibility is an important characteristic of liveness. It is related to creating *the sense of being there*, typically when it is not, thus the spectators can see, hear, and feel the creative process. Therefore being co-located in real-time does create a natural perceptibility.

One underlying assumption that the perceptibility have is that spectators should be able to access and understand the creation process from their perception. Therefore, the perceptibility should include various concepts from the low level human sensory system limits to the high level abstraction, such as clarity of interaction. What constitutes sufficient perceptibility can change depending on the context, creators' intentions, sensory systems that are available (or targeted) and the socio-technical factors on the creative process. For example, familiar creative activities, such as painting or playing a violin solo, can have different levels of perceptibility from other rather unfamiliar activities such as programming or DJing in a club because the level of background knowledge that the spectators have on two different activities can differ. Research in live electronic music, in which performers often sit behinds a laptop computer and an audience is not aware of what is going on, highlights the challenge of delivering liveness that is decoupled from a performer's physical actions [41, 42]. In addition, the notion of perceptibility can dynamically change over time with the emergence of new media. Virtual reality can be used to create audiovisual perceptibility and many researchers are working on supplementing the other senses as well [43, 44, 45]. Revisiting the example of film, a film may bring more liveness by augmenting the sense of being there, not only through well-made content, but also by media technologies, such as a surround screen[46], 3D cinema [47], spatialized audio [48], and haptic feedback [49]. The technological space of enhancing perceptibility is dynamically expanding and challenging to define. In this dissertation, I focus our discussion on the first two values: immediacy and continuity. These two temporal dimensions — immediacy and continuity — provide metrics for us to evaluate and compare interactive systems in terms of liveness.

1.4 Live Collaborative Creation with End Users

The systems presented in this dissertation support *live collaborative creation*, whereby end users can collaborate with creators to create an artifact in real time. Note that there can be multiple kinds of live collaborative creation, as it can simply refer to the collaborative process in live settings, where an artifact in creation is perceptible in real time. For example, two musicians collaboratively improvising in front of an audience can be seen as a form of live collaborative creation in which users — the audience in this case — are not involved.

Only the creators of the artifacts are collaborating in real time. In this dissertation, I study a type of live collaborative creation where the users are not only situated in the live process of creating an artifact, but also able to participate in the process, collaborating with the creators. Examples of such live collaborative creation are suggested in Figure 1.2. Live collaborative creation is an effective way to extend the benefits of live settings by bringing users closer to the creation process. This is also beneficial for creators, as they can effectively satisfy their users. This is because creators can communicate with users and get immediate feedback on the artifact in progress. The improved effectiveness of user co-creation in product development and service design was confirmed in an asynchronous setting through the closer fit of products to user needs and higher commercial potential [50, 51].



Figure 1.2: Examples of live creation: In teppan-yaki restaurants, customers can see the process of creating an artifact (top left). At concerts, audience members can see musicians performing in front of them in real time (bottom left). In some restaurants, customers can actively participate in the cooking process by choosing ingredients (top right). Some music is composed to incorporate audience participation, such as Queen’s “We Will Rock You”, in which audiences are guided to stomp and to clap to the rhythm (bottom right).

Kaulio suggested a framework for the analysis of methods for customer involvement in product development [52]. The framework includes two dimensions: the longitudinal dimension which represents the time point at which users (customers) are involved in

a product development process, and the lateral dimension, which captures how much customers are involved in the design process. Live collaboration creation occurs at the final stage of creating and delivering an artifact to users. Their involvement is significant as participants directly contributes to the creation process and make the artifact.

1.5 Problem Statement and Thesis Statement

Live creation — revealing the process of creating artifacts to users in real time — and live collaborative creation — collaborating with end users to create artifacts in real time — pose many challenges and certainly are not suitable for all kinds of creative activities. In both cases, revealing the creation process can yield more cost to both parties - creators and users - and can be against to the goal of minimizing the transaction cost if the live process does not add value to the exchange [53]. Therefore the tension between the cost involved in and the value added with the live process is the key factor in determining whether to reveal creative process live or not.

Regarding the cost of live creation and live collaborative creation from the users stand point, it can be costly being there for live creation process takes time and the additional time that is required to perceive the process. Certainly, examples of live creation and collaboration that we have seen occur in the short time span, typically on the order of minutes and hours at most. This is to minimize the time that it takes for the audience. This time consuming nature of live creation compels the creators to carefully determine the effective boundary between the creative process that should be revealed live and the pre-production steps that can be prepared prior to the live process in order to minimize the time. For example, preparing ingredients for cooking — purchasing, storing, and washing them — is not part of live cooking process as it can significantly extend the time. In [54], conference attendees were precisely guided to build a large-scale architecture in which most of the components, the design, and the guided instruction systems were planned and prepared in advance for the live collaborative creation. To this end, creators can have additional cost in preparing the live process for making it more informative, performative, comprehensive, and risk-tolerant.

On the other hand, the value added by revealing the process of creation in real time can vary depending on the artifact that is created. In general, the artifact of which its value increases with recency — minimizing the delay between creation and consumption — can have potential increase in its value with live creation. In some cases, the recency itself can have a value in its quality — e.g., fresh cooked vs. microwaved food. The live requirement can result from a highly personalized nature of artifacts usage, services, or content — e.g., mass-produced ready-made products(ready-made clothes) vs. made-to-order products(order-

made dress). For certain types of contents, the consumption activity inherently take time — i.e., watching a movie, listening to music. In such cases, the value of live performance can originate from the value of uncertainty involved in the live process — e.g., live broadcast of sports events vs. recording of them given the final result is known — and the creators' efforts to reconcile such risks — e.g., live music performance vs. videotaped performance in a studio which could have done in multiple takes. Similarly, liveness can be more appreciated for the artifacts that can be presented live. For example, students taking massive open online courses(MOOC) were engaged more with lecture videos that are similar to a live lecture [55]. Lastly, the interactive components that can be used to engage the target users and to collect immediate feedback have created new live digital media such as Twitch, Facebook Live, and Periscope [27, 22].

Many of these qualities present in live collaborative creation may affect our interactions with technology for many reasons. Due to its temporary nature, creators can neither expect a certain structure from users nor choose those who will participate in advance of the collaboration. In addition, such a collaboration should occur in a short span of time. Therefore, computational systems must quickly scaffold a collaborative environment in which any user can participate in the creation process.

In addition, there are two disparate groups of stakeholders — creators and users — in live collaborative creation. In most of the setup, users are the ones whose needs must be met. Users' satisfaction can be a composite of their experience in the creation process and the artifact that is delivered. The roles and goals of two groups differ, and their interaction with the system should thus also be designed separately. Therefore, commodity real-time groupware and communication tools may not be sufficient. Also, there often exist differences in expertise between the two groups, one being expert and the other being a novice with no domain expertise. Therefore, computational systems for live collaborative creation need to be designed carefully for the broader population to be able to participate in the creation process.

Lastly, some types of creation are simply not practical to perform in live settings. For example, it is difficult to imagine writing a book live, which may take months and years, in front of readers. However, the question is how we can replicate the benefits of involving users in the creative process when the artifact itself cannot practically have liveness. It is common practice in some domains, such as media production and product design, to document the creation process of artifacts, typically in the form of films, and share them with users so that they can better understand the artifact. We wish to extend this approach to general domains and design the artifact to contain the liveness in the creation process.

This dissertation work seeks to evaluate the following thesis statement:



Figure 1.3: An artifact is typically consumed long after the time it was created. My research explores live collaborative creation in which users can be involved in the creative process. There are three subgoals that I present: 1) to identify and address the challenges in computationally supporting live collaborative creation, 2) to develop tools and methods that can reduce the barriers of entry to live collaborative creation for non-experts and broaden the applications of live collaborative creation, and 3) to bring liveness into asynchronous communication when we cannot collaborate live.

Interactive systems can coordinate and mediate live collaborative creation between creators and end users for more immediate, engaging, and direct effects of user involvement.

To explore this statement, my dissertation addresses three research questions.

- **(RQ1)** What are some of the challenges in supporting live collaborative creation between creators and users, and how do we address them?
- **(RQ2)** How can we lower the barriers to live collaborative creation for non-experts?
- **(RQ3)** How can we bring liveness into asynchronous communication and collaboration?

Each research question is depicted in Figure 1.3. The following sections will briefly introduce the approaches I have taken to each research question.

1.6 Identifying and Addressing Challenges in Live Collaboration

I have worked on a thread of research that seeks to create novel modes of live collaborative creation for creative and artistic domains. I chose two instances of collaborative tasks, which are typically performed asynchronously, and developed computational system that support them in live settings to identify and to address general challenges that users and creators face in live collaborative creation.

1.6.1 Qualitative User Study on Real-time Collaboration between an Expert and Novices in Crowdsourcing

An established approach to crowdsourcing involves breaking one large task into smaller microtasks that non-expert crowdworkers can solve independently, without the need to communicate directly with the requester. However, microtasking fails for creative tasks, as steps into which the task can be decomposed are not well defined; additionally, subtasks are often interdependent. Real-time collaboration between requesters and crowdworkers can address these challenges. This type of continuous interaction is increasingly frequent, and is broadening the applications of crowdsourcing; currently, though, we lack a deep understanding of interactivity between requesters and workers. To better understand how requesters speak to collaborate with crowdworkers in this setting, I conducted a qualitative user study with a crowd-powered tool for user interface prototyping where a requester (a user) asks crowdworkers (creators) to create a user interface prototype (an artifact) [56].

The results revealed challenges that requesters face when collaborating with crowdworkers. Following is summarized findings from the study.

- Requesters actively collaborate with workers and understand the benefits of working with them in real time.
- Expert requesters may speak in a style that makes it difficult for crowd workers to comprehend their requests (less descriptive, more jargon).
- The asymmetry in communication modalities (speech vs. text) causes confusion.
- Simply sharing the visual context was not enough to effectively coordinate collaboration.
- Speech pace varies among individuals; rapid speech may result in a backlog of requests.

These findings are not limited to the context of UI design, and are immediately applicable to systems that employ a collaborative setting in which a requester and workers directly communicate to work on a shared artifact. I suggested design implications and tools for similar interactive systems, such as real-time groupware and live social media that has similar asymmetry in communication (e.g. Twitch, Facebook Live).

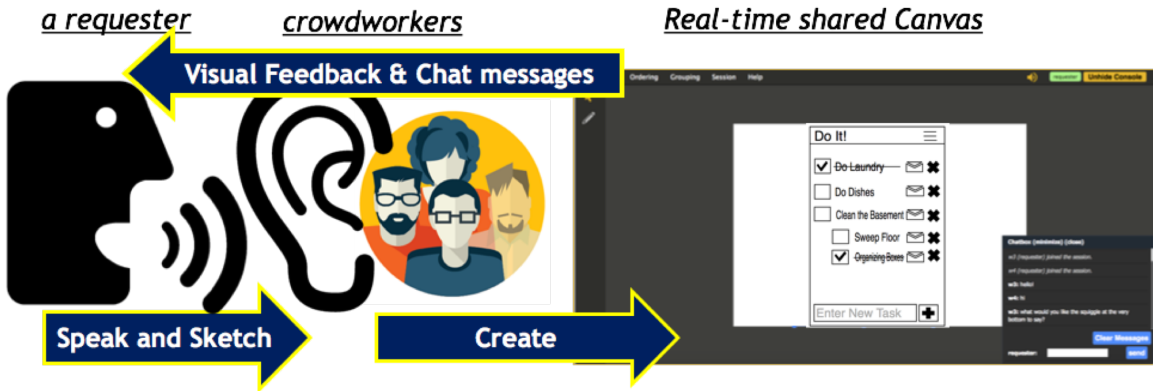


Figure 1.4: Real-time collaboration between a requester and crowdworkers for a GUI sketch prototype: 1) A requester verbally describes and draws a rough sketch on a shared drawing tool in real time. 2) A set of crowdworkers listens to the verbal description with the rough sketch. 3) The crowdworkers enhance the rough sketch into a prototype with higher fidelity, and communicate with the requester through the chat interface.

1.6.2 Environments for Live Collaborative Programming

Programming is a complex activity for real-time collaboration that needs to be highly coordinated and can have many conflicts on the fly. In this chapter, we show how artists' experimental takes on musical aesthetics pose intriguing computational and engineering challenges if live collaborative programming for their artistic expression. In the experimental music performance *Live-Coding the Mobile Musical Instrument*, two programmers (creators) collaborate to write a mobile music instrument (an artifact) on a tablet live on stage, while a musician (a user) performs on the musical instrument that is being developed over a wireless network [57]. In this extreme setting, a pair of programmers develops a single interactive program — a mobile music instrument — in real time while a musician plays it. The programmers need to have a clear understanding of the live state of the running program as it is being changed by their collaborators' code, as well as how the musician interacts with the program; these challenges were not adequately addressed by pre-existing tools. From the user's perspective, the instrument's usability abruptly changes as new code is executed. To address this challenge, I developed a programming environment that shows the program state in real time, including how the user interacts with the program, within the code editor [58]. In addition, the programmers can freely improvise without worrying about naming conflicts in declaring variables and functions, as each programmer has their own namespace. They can also share objects (e.g. variables and functions) selectively, should the programmers wish to work with some objects collaboratively. To minimize interference caused by changes in the program's state during the performance, the programmers can let

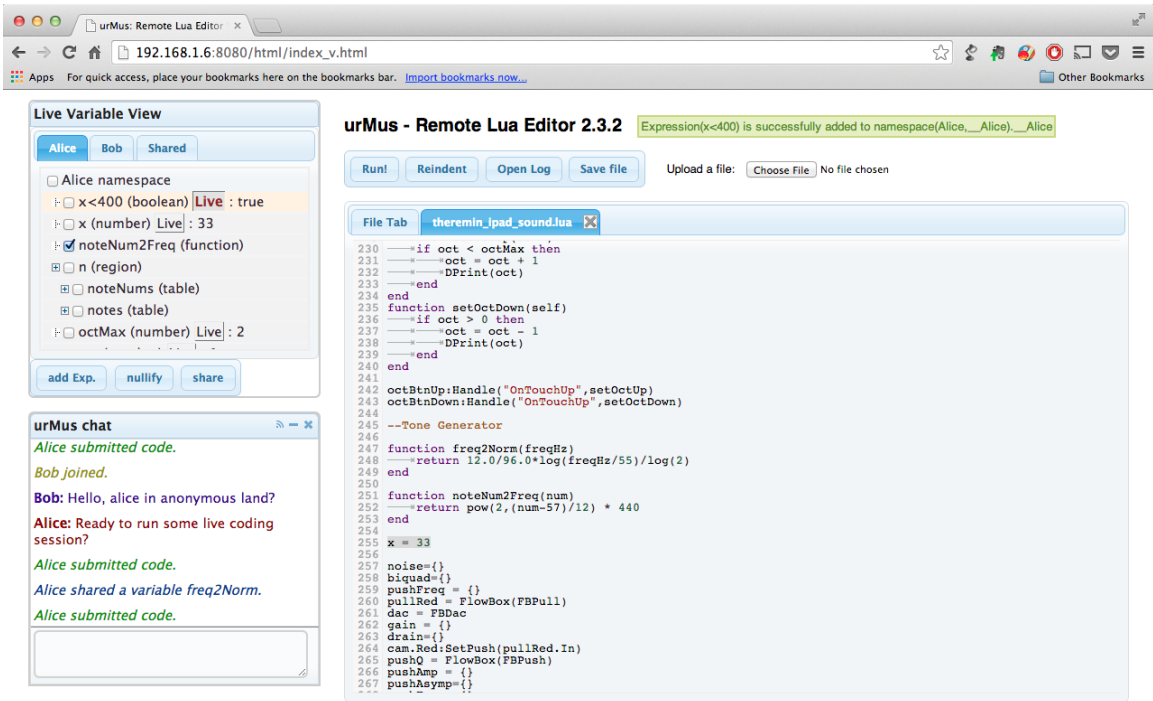


Figure 1.5: A programming environment for collaborative, live coding. The live variable view in the top left corner makes the program state visible to all connected programmers in real time. Note that each programmer has a unique namespace to avoid spontaneous naming conflicts between variables and functions.

the user control the timing of updates to the program. The exploration of this experimental case encompasses and addresses the challenges of potential applications of collaborative programming in real time: the program’s state is visible to the programmers without the potential risks of breaking each other’s code. I suggested a similar programming environment for crowdsourced software engineering, wherein expert programmers can self-coordinate and be aware of their collaborators’ work live [59].

1.7 Lowering the Barriers to Live Collaborative Creation

Taking this approach a step further, I have explored ways to include non-experts in the creation of artifacts that require domain expertise through live collaborative settings. Specifically, I have developed interactive systems that allow non-experts to quickly contribute to the creation process within minutes. In particular, the systems allows non-experts to create an interactive UI prototype and to participate in an interactive music performance.

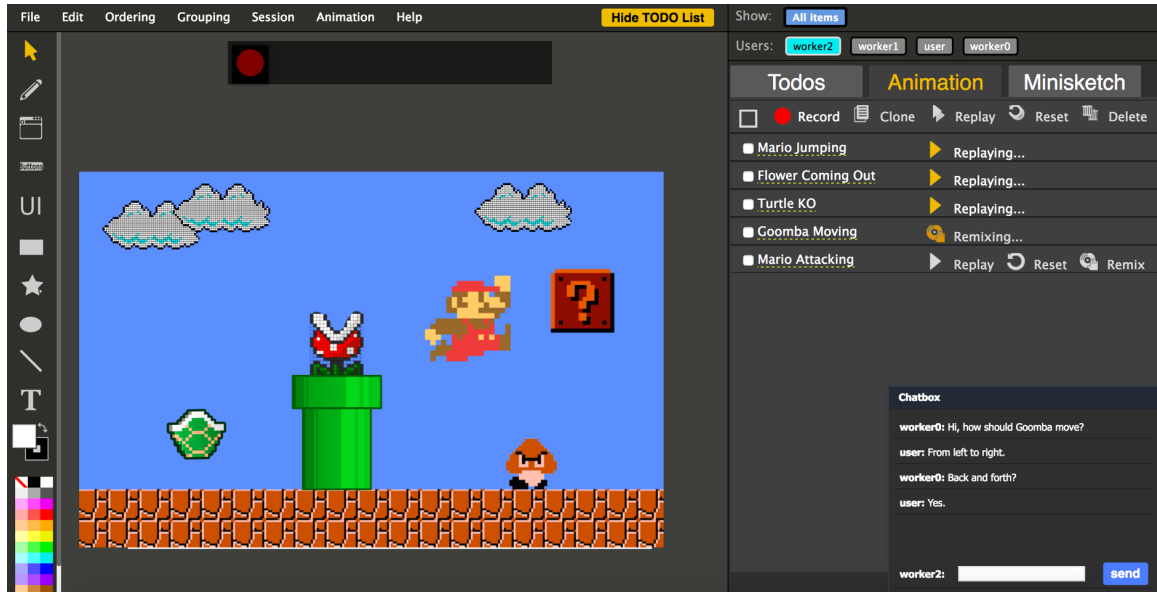


Figure 1.6: *SketchExpress*: Non-expert crowd workers can create interactive behaviors in a few minutes using the demonstrate-remix-replay method. The final sketch will have a set of animations that demonstrate multiple behaviors.

1.7.1 *SketchExpress*: Remixing Animations for Crowd-powered Prototyping of Interactive interfaces

I created *SketchExpress*, a crowd-powered prototyping tool to augment early sketches with interactive behaviors created by non-expert crowdworkers [60]. In this crowd-powered system, requesters verbally describe desired interactive UI behaviors while crowdworkers collaboratively demonstrate the behaviors in real time, based on the given description (Figure 1.6). However, manual demonstration is problematic: some behaviors, for example, are too complex to demonstrate manually, and the demonstrated behaviors are ephemeral. To resolve these challenges, *SketchExpress* provides a novel interaction method that lets crowdworkers demonstrate these behaviors and remix their demonstrations, preserving the interactivity of the manual demonstration and improving the fidelity of the animation. The generated behaviors can be replayed by UI designers. *SketchExpress* allows workers to create a complex animation within 2.7 minutes on average, with a 27.3% increase in quality (recall) compared to the manual demonstration. Once created, the artifact (sketch) contains expressive, reproducible behaviors; users can interact with the sketch with a single click. The system mitigates the challenge of real-time collaboration by incorporating asynchronous interaction techniques, such as record-and-replay and remixing (post-processing), which alleviates the real-time pressure of manual demonstration.

1.7.2 *Crowd in C*: Audience Participation in a Musical Performance Using Smartphones and Cloud Computing

Musicians have long sought to create musical performances which involve an audience as part of the music-making process, as this is an effective way of engaging the audience. For example, in popular music, musicians induce audience participation by making sounds directly; e.g., by singing, clapping, or stomping feet (as in Queen’s “We Will Rock You”). Audience participation is not only a method for musical performance, but can be expanded to different contexts, like theatrical arts, classrooms, and public events. I have developed computational systems for audience-participation music pieces in which an audience is the only musician, collectively generating sound for a music piece at a concert hall using smartphones [61, 62]. There are several requirements in creating a large-scale participatory system through which audience members can perform a piece of music: the system needs to be immediately accessible to non-experts, the system needs to sustain the audience’s interest in participation for the designated period, and the system needs to allow musicians to orchestrate the audience members to perform a musical piece without interrupting their participation. The system that I developed for the piece *Crowd in C[loud]* provides audience members with an ad-hoc social network within the musical instrument, using the metaphor of online dating [62]. Each individual’s short composition, which is looped, serves as a profile, and participants browse through each other’s profiles (Figure 1.7). This allows audience members to socially interact with one another and experience a process similar to how musicians explore various musical ideas, instantly facilitating collaborative music making at public events. Through large-scale audience participation, hundreds of short tunes played on smartphones can create an ambient sound in a predetermined chord (C major, for example) and a musician on stage can remotely change the mapping (i.e. range of pitches) of the musical instrument to follow the chord progression of the composition. The key component that mitigates the challenges of participating in a live music performance is the asynchronous interaction of using social media in participation, as opposed to playing musical instruments, a time-sensitive activity requiring a high degree of synchronization with other musicians.

1.8 Liveness for Asynchronous Communication and Collaboration

In the systems described above, the real-time visibility of the creative process to users constitutes liveness. However, how can we preserve the benefits of liveness when we

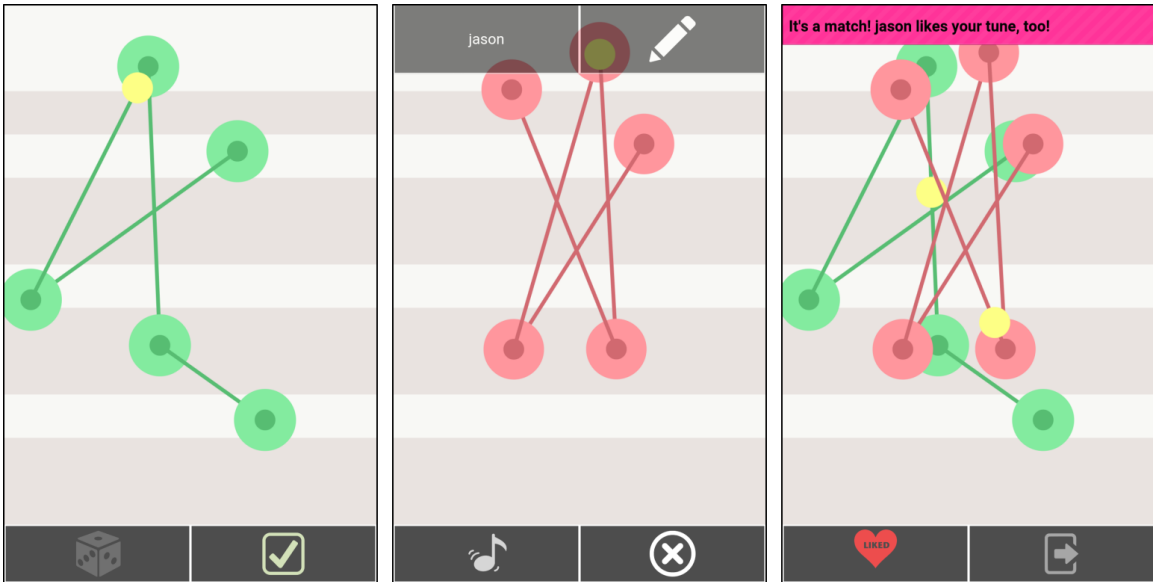


Figure 1.7: A web-based musical instrument for the audience participation music piece *Crowd in C[loud]*. The short composition (left) serves as a profile on a social network, and users can browse other people’s profiles (middle) and play together (right). The social interaction among audience members helps sustain their interest for the duration of the performance.

cannot present the creation process to users in real time? I searched for answers to this question through systems that preserve liveness in asynchronous collaboration and written communication. Writing is an expressive process guided and adapted by thoughts that evolve over time. The dynamic process of creating a written artifact (a document or program code) can be informative and expressive, and the real-time history of a written work contains information not available in a static copy. For example, watching an instructor live-coding a program in a classroom reveals the temporal order of how the program is constructed and reflects the programmer’s dynamic thought process, which the instructor wishes to convey. By contrast, existing methods of archival, such as file saving, source version control, and even undo stack trees, do not archive the real-time history of how a written artifact has evolved. To that end, I introduce two systems that capture liveness in asynchronous collaboration settings, particularly for programming and writing tasks.

1.8.1 *CodeOn*: On-demand Programming Assistance

I explored the possibility of preserving liveness through asynchronous collaboration in a programming context. *CodeOn* implements an asynchronous, on-demand support system [63]. In *CodeOn*, programmers can ask programming questions to expert crowdworkers

Codeon

Request list

- Request 42 (new comment)
- Request 41
- Request 40 (new title)
- Request 39
- Request 38
- Request 37
- Request 36

Close Discussion

Helper : Hi, I want to clarify one thing. I realize you have a countdown. I guess I need refresh after three seconds? (44 minutes ago)

requester : Yes, I forgot to mention that. Thanks for pointing that out!! :) (43 minutes ago)

Discuss request with requester ...

```

audience.js
571
572 // Set the name of the next div
573 - function setNextDivName(divName) {
574   var actualTindered = document.getElementById('tindered');
575   actualTindered.innerHTML = "";
576   actualTindered.appendChild(document.createTextNode(divName));
577 }
578
579
580 - window.onbeforeunload = function(){
581   return "";
582 };
583
584 - /***** HELPER : My code starts here *****/
585 - window.onbeforeunload = function() {
586   var errorMsg = "this is the text message that will appear on the prompt window."
587   return errorMsg;
588 };
589 - /***** HELPER : My code ends here *****/
590
591 - function randomizeNote(){
592
593   for (var i=0; i< patternSize; i++){
594     var note = new Note();
595     note.setPosition(Math.random(), Math.random())
596     pattern[i] = note;
597   }
598
599   for (var i=0; i< patternSize-1; i++){
600     pattern[i].distance = dist(pattern[i].x * w,pattern[i].y* h,pattern[i+1].x* w,pat
601   }
602 }
603
604 - /***** HELPER : My code starts here *****/
605 - function refresh(){
606   window.onbeforeunload = null;
607   showMessage("Info", "This page will be refreshed in 3 seconds...", true, 2500);
608   setTimeout(function(){
609     window.location.reload();
610   },3000);
611 }

```

an hour ago

Q: prompt an warning when a user leaves/refresh an webpage.

I want to make sure that a user have a chance to get notified when the content is not yet submitted. Also when submitted, how can I let user leave the page without an alert?

Status: pending

▶ Play
Original ▾
Respond
Git Pull

Code Annotation

Annotation Response List:

you can set onbeforeunload to null and it will de-register the alert callback. an hour ago

set this variable to whatever msg you want. 31 minutes ago

Explanation:

Have an event handler bound to onbeforeunload event so that you can detect when the user is browsing away from the current page. onbeforeunload is not consistent across browsers (I don't think Opera responds to it IIRC, but have no way to currently test).

Figure 1.8: Codeon: the code context that is associated with a programming question is visible to an expert crowdworker.

without leaving their integrated development environment (IDE), and the question generated includes a replay of the question being asked and the code highlighted during the speech act, simulating in-person communication in a pair programming session. While the collaboration between programmer and crowdworker is asynchronous, the dynamic context of asking a question within the IDE is preserved and transferred to the response editing system. Our in-lab user study showed that the system reduced time and effort in seeking help for programming questions by 70% compared to state-of-the-art tools.

1.8.2 *Live Writing*: Accessing Real-time History via Record-and-replay in Writing

In my *Live Writing* project, I developed a web application that extends existing web-based text editors. The application records writing activity at a high resolution of detail, such that the writing process can be reproduced in real time and replayed later (Figure 1.9) [64]. In the previous example of live coding in a classroom, the *Live Writing* system allows instructors to create such a replay, which can be shared asynchronously with students via a link to a web page which will replay the live coding session in real time. Furthermore, the system can play an important role in the writing process for writers themselves, enabling them to recover context when resuming a writing task or when writing collaboratively. I have explored the effects of *Live Writing* in supporting collaboration in live coding [64] and in academic writing [65]. Currently, the application is deployed publicly. (demo: <http://echobin.com>). The idea of *Live Writing* has been explored as a form of performing art, in which a musician performs an audiovisual piece by writing a poem live on stage in a web-based editor I developed [66]. A visualization technique is developed to deform typography dynamically in response to any real-time data; e.g., audio or sensor readings, using GLSL shader language (Figure 1.10) [67].

1.9 Contributions and Outline

This dissertation will make the following contributions per each research question:

- Research Question 1 : Identifying and addressing challenges in live collaborative creation
 - Chapter 3: findings and design implications from a qualitative user study on live collaboration between a requester and crowdworkers

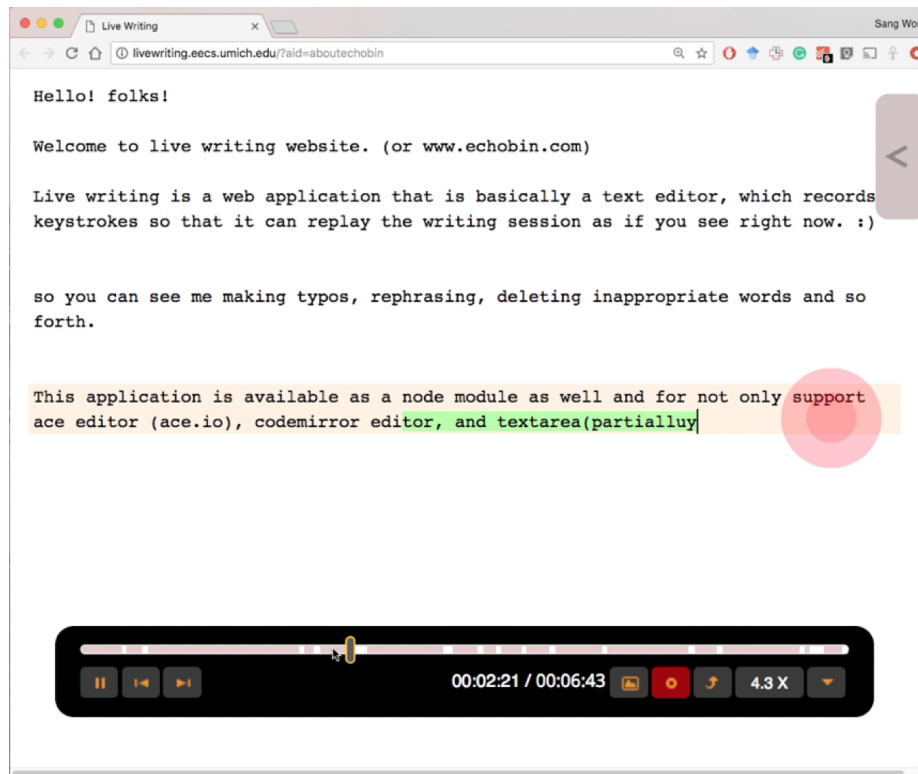


Figure 1.9: Live Writing: A web-based text editor that can record and replay writing activity in real time. This type of screencast preserves textual information that users can use.



Figure 1.10: Live Writing: Gloomy Streets. Writing a poem on stage live in front of the audience becomes an audiovisual performance. The piece uses temporal typography to enhance communication with the audience.

- Chapter 4: a programming environment for collaborative, live programming with a shared program state
- Research Question 2 : Lowering the barriers of live collaborative creation for non-expert end users
 - Chapter 5: a crowdpowered GUI sketch tool for prototyping interactive behaviors within a few minutes
 - Chapter 6: a live music performance system for large-scale audience participation at concerts, using a mobile ad-hoc network(MANET)
- Research Question 3 : Capturing liveness in asynchronous communication and collaboration.
 - Chapter 7: a programming support tool for on-demand assistance from expert crowdworkers
 - Chapter 8: a writing environment that records and replays writing activities in real time for context recovery and temporal expression

CHAPTER 2

Related Work¹

This research draws ideas from real-time collaboration in various domains. In particular, live collaborative creation is based on the previous works of liveness in programming environments, real-time groupware and real-time collaboration in crowdsourcing. This chapter summarizes related work in these areas.

2.1 Liveness in Programming Environments

Liveness has been used in programming languages to indicate the different levels of delay between programming activity and the programming state revised by the programming activity [70]. Tanimoto suggest four different levels of liveness in programming environment [71]. Typically the visibility of program outcome is separated by the build-compile-run cycle (level 2). However, some programming environment accomplish the level 4 liveness, in which the program state is updated as a programmer changes their code [72]. In this dissertation, the definition of liveness has been extended and simplified (as a binary classification) to cover broader applications in interactive system. The liveness indicates the real-time visibility of an artifact to consumers. In the case of the programming, the artifact is the program state which needs to be visible in real-time when a programmer, who is a consumer of the artifact as well as creator, writes code.

The notion of liveness used in programming environments does not conflict with the definition that I suggested in this dissertation. In fact, the definition of liveness used in this thesis originates from and is carefully chosen to include the liveness used in the context of programming. As aforementioned in Chapter 1.2, it has four components that are common in live setting: 1) an artifact : a computer program that a programmer is writing, 2) those who create an artifact : the programmer, 3) those who perceive it : the programmer, 4) and the concurrence between 2) and 3) as it is typically the same person, who creates something

¹Portions of this chapter appear in [60, 56, 68, 69] with the permission of Antonio Deusany De Carvalho Junior, the lead author of [69].

in action with coding and who perceive the creative process and check out the result, which is program state. The definition of liveness is immediate and continuous visibility on the creative process and the artifact. Based on the Tanimoto's four levels of liveness, Level 4 liveness implements immediate and continuous visibility on the program state according to the programmer's activity (keystroke, for example). The other three levels (level 1, 2, and 3) of liveness has non-immediate and discontinuous visibility. Liveness beyond the programming environment is already discussed in Chapter 1.3.

2.2 Preserving Liveness in Groupware

2.2.1 Timeline and Replay in Programming and Writing

The playback of programming and writing has been explored for various purposes including the change awareness for collaboration. Most notably, replay has been suggested as a design implication of programming environment to assist resuming interrupted programming tasks [73]. In practice, providing a chronological history of activities outperformed note taking strategy for task resumption and was the most preferred by developers [74]. Replay the rescue implements such playback function within the integrated development environments (IDE), maintaining snapshots of code (finer than user commits) to keep track of software evolution [75] The system outperforms version control system on helping developers to answer common questions related to software evolution in a collaborative context. OperationSliceReplayer records edit history in a programming environment and eliminates non-essential editing operations to extract snapshots. Azurite, on the other hand, implements the visualization of a fine-grained history of code where edit history of code is visualized as a color-coded timeline, and a programmer can compare two snapshots, search history information of highlighted code region or execute selective undo [76]. CodeSkimmer simulate video skimming like interface (play, fast-forward, backward) toolbar in the programming environment. [77]. Many of the previous works explore replay in the programming environment, the focus was not the real-time replay, but more on the navigating code history back in time. Further, few of them include a formal user study that investigates how such replay helps in text comprehension and context recovery in the collaborative setup.

Visualizing a history of a document has been active research topic especially with the rise of Web 2.0 such as Wikipedia [78, 79]. However, the real-time or fine-grained replay of a collaborative document is less explored. The replay of wiki edits has been visualized with highlighting and animation, showing the effectiveness in gaze following [80]. Logging fine-grained history of keystrokes is a common approach in the field of writing research

to analyze real-time writing behavior. The writing researchers have developed numerous keystroke logging applications as it provides a non-intrusive and inexpensive technique to monitor user inputs. Inputlog[81] and ScriptLog[82] are such programs used in context of writing research². Most keystroke logging applications include real-time playback recorded keystrokes and it is an effective approach to help subjects account for their writing in retrospect, which is less intrusive than having them think aloud while writing [83]. These applications are mainly for the research purpose of real-time writing analysis rather than for writing to assist collaborative writing.

2.2.2 Asynchronous Change Awareness

Awareness is the ability to understand the activities of others and required to coordinate collaboration, usually in the systems that are distributed or asynchronous [84]. The awareness provides an important foundation for realizing liveness in computational systems as the immediate and continuous perceptibility can be seen as an extreme version of awareness. Therefore, my work in preserving liveness relates to the current body of works in developing awareness in groupware. The notion of asynchronous change awareness has been used to represent the capability to recognize and describe changes made by participants in a collaborative project [85]. The nature of asynchronous collaboration often let users miss important changes, and such failures may cause significant cost [85]. Particularly in software engineering, maintaining awareness was found to be the most frequent sought information [86]

The change awareness has been realized in different forms in various domains from graphical applications to collaborative writing. Typical change awareness functions in a groupware inform users about changes made users by highlighting them and tracking history of documents. The placement and the granularity of such change awareness have been investigated in unified modeling language (UML) diagram [87]. Many modern word processors implement tracking changes of all edits made, which are desirable by users in the context of collaborative writing [88]. GraphDiaries relies on animated transition for graph visualization, showing that it outperforms existing techniques both in time and error rate [89]. The effect of highlighting and animation to navigator history of a collaborative document was investigated in context of Wiki document, implementing timeline playback of edit history with animated transition [80] One of the issues of realizing change awareness is that the change is derived based on the series of snapshots that are created by a user's will (e.g. file save or code commits). Such coarse-grained history can cause the information loss,

²See http://www.writingpro.eu/logging_programs.php for more complete list of keystroke logging programs.

and researchers addressed the issue by having finer-grained history within the software [90]. In this work, we record user activity in the finest level possible and turn the logs to generate real-time replay for asynchronous change awareness.

2.2.3 Networked Live Coding for Collaboration

The notion of liveness is directly inspired by the emerging field of live coding [91] in computer music and creative coding. Live coding is audiovisual performance where a programmer writes a program live in front of an audience on stage. The outcome of the program is generative music and perhaps visuals. In live coding music performance, the programming language can be viewed as a musical instrument [92]. It is typical to project program code text in the performance space for the audience to understand the process of music making. This highly visible nature of the coding process makes the performance aspect of code writing explicit to the audience. This principle is well captured in the following statement of *TOPLAP*³ (live coding community) manifesto; "Obscurantism is dangerous. Show us your screens." Modern computers all but invite the use of networks in collaborative music-making contexts. The same can be said for collaborative live coding. Networked communication and data sharing can facilitate collaboration among live coders. While collaborative live coding does not necessarily mean that computers need to be connected over a network, the potential of networked live coding has been present from live coding's inception of live coding [91]. Realizing networked live coding requires detailed consideration of the networked system. There are a multitude of design choices that pertain to what kind of data is shared and how the system maintains that shared data across different network topologies and how it will facilitate collaboration among musicians over the network. In particular we will review time sharing (synchronization), code sharing (text/program representation), program state sharing (run-time states, variables, objects, memory), access control, and communication facilitation (chat). We hope this classification provides an overview for live coding researchers creating a collaborative live coding environment and that it sheds light on collaborative aspects supported by the system.

Time sharing: In any form of multi-performer music, what is important is playing together, that is to say being synchronized. This is certainly also true for live coding performance. Hence, in a networked live coding setting, it is important to consider how to facilitate synchronization. One crucial precondition to this is having one synchronized clock between machines. Many networked live coding environments enable clock synchronization or implement ways to synchronize timing of musical events [58, 91, 93, 94, 95, 96, 97]. The

³<http://www.toplap.org/>

method of clock synchronization varies depending on the architecture of the network and the distributed nature of the ensemble and well-known methods can be found in [95, 98]

Code sharing: Some live coders share actual code fragments among members of ensemble during the performance. Live coding environments on a web browser such as *Gibber* [99] or *Sketchpad* [100] offer a Google doc-like shared editor. In *LOLC*, commands are typed into an instant messaging-style interface that shows both commands and chat messages so that you can see your code, others' codes and chat messages [101]. Sharing code text is essential not only in facilitating collaboration but also in improving the sense of collaboration. For the same reason that live coding is projected on screen to communicate with an audience [102], participants had a more engaging experience monitoring the progress of collaborators.

Program-state-sharing: Instead of sharing code text, some live coding environments enable sharing dynamic objects or variables in the program state. The ways in which sharing objects is implemented are diverse: re-rendering objects by evaluating code fragments in both the local machine and the remote machines [93, 103, 104], transmitting sound objects as serializable data [94], synchronizing the value of variables using tuple space [96], or being shared inherently due to one centralized program state for multiple live coders [58]. One may think that sharing code text is good enough as one has access to the code that can reproduce the same objects in a local machine. For a live coder, however, reading, interpreting, and evaluating the code fragment are additional cognitive loads. Furthermore, the code text in the editor is not a complete representation of the program state and there almost always exists a discrepancy between the code text (*State of Code*) and the program state (*State of World*) [105]. In other words, the code associated with a certain sound (or any outcome from the live coding) at the moment may not even exist in the text editor any more because it may have been modified.

Access-control: Regarding shared code text and state synchronization, it should be noted that the nature of collaboration will vary depending on the level of permission (e.g., read/write/execute) given each live coder to the shared data. One can design a system to allow all types of permission for all participants, which enables open collaboration. This would be like the early networked music piece *The Hub's Borrowing and Stealing* [106]. In [94], the environment supports the sharing of musical patterns with read/execute permission so that shared objects are not mutable by other live coders. Or a live coding environment can select a more conservative strategy where a live coder can choose to share a certain set of variables in the selective manner [58] and choose to evaluate a certain code fragment remotely to obtain a dislocated sound [103].

Communication-facilitation: Enabling chat is an effective strategy to facilitate communication between live coders as well as to engage an audience in the communication loop

when projected on screen [58, 94, 99, 100, 104, 107]. Lastly, we want to point out that sharing actual low-level outcome (such as raw audio) has been less explored.

2.3 Collaboration in Crowdsourcing

Crowdsourcing for human computation engages people through an open call to contribute to a computational process in order to solve problems that machines cannot solve alone. Crowd workers can be recruited on demand from platforms like Amazon Mechanical Turk [108]. These workers are often quasi-anonymous to requesters [109], and have unknown and varied skills/experiences. In crowdsourcing, work is often coordinated by dividing it into small, context-free units called “microtasks” [110]. Significant effort is required to generate microtasks and aggregate responses. While microtasks have been shown to be useful for tasks that have a clear goal and an established problem-solving process, open-ended tasks (e.g., designing a UI) are better solved through collaboration between requesters and workers [63].

However, tasks that do not have a fixed process and require continuous involvement, like designing a UI prototype, have been under-studied. An early crowd-powered system that handled open-ended tasks was Soylent, which provided document-editing assistance, such as proofreading or text shortening with a general process: Find-Fix-Verify [111]. Research has continued to find new ways to coordinate workers on writing tasks for academic writing [112], creative writing [113, 114], and on community resources like Wikipedia [115].

Broadly, crowdsourcing can be seen as an asynchronous collaboration between requesters and online crowd workers that occurs over a long period of time (e.g., days or months). The most established approach to crowdsourcing involves breaking a task down into microtasks that non-expert crowd workers can solve independently [110]. This method is effective in settings where there exists a definable problem-solving process and immediate feedback is not necessary. However, microtasking has limitations in addressing various ‘complex’ tasks [116]. We address the problem by enabling real-time continuous interaction between a requester and crowdworkers.

2.3.1 Real-time Collaboration in Crowdsourcing

Continuous real-time crowdsourcing [117] can address this challenge since it not only elicits rapid responses but also enables requester-worker interactions [118]. VizWiz [119] showed that the crowd could answer visual questions in under a minute. Legion [117] introduced continuous real-time crowdsourcing for collective control tasks, and Adrenaline [120] used

a “retainer model” to bring and direct crowds to a task in seconds. LegionTools [121] builds on this idea, and is the first publicly-released tool for recruiting and managing real-time crowds. EURECA [122] leverages online crowds which collaborate to help robots robustly identify 3D point cloud segments corresponding to user-referenced objects in near real-time. Real-time crowdsourcing allows requesters to seek help on demand and get assistance quickly. Researchers have explored tools and methods to enable the convenience of various tasks. The retainer model enabled the recruitment of crowd workers in only a few seconds [123]. Coordinating and aggregating crowd workers’ responses in real time allows workers to control graphical user interfaces [124]. Chorus, an intelligent conversation assistant, leverages crowd workers’ efforts by having them answer any question a requester asks in an instant messenger [125, 126]. Recently, researchers proposed an augmented whiteboard that allows a co-located team and remote crowd workers to work together in a brainstorming session [127]. It is shown that co-located teamwork among workers enhances their collaborative experience in comparison to remote independent crowdsourcing [128]. We believe that real-time crowdsourcing can improve task outcomes by enabling a fluid, iterative workflow where requesters and workers can quickly provide feedback to one another.

2.3.2 Facilitating Communication in Crowdsourcing

Facilitating frequent communication between a requester and crowd workers is an effective way to improve the outcome of crowdsourcing. For example, the literature shows that getting feedback on a task outcome encouraged the workers to revise their answers and yielded a better final outcome [129, 130, 131]. Similarly, a user study in VizWiz shows that requesters preferred continuous interaction with crowd workers [119] in an accessibility context. Chorus:View demonstrates the same benefit of continuous communication [118].

Researchers have developed systems and methods for users to effectively communicate with crowd workers and for crowd workers to collaborate with each other. WearWrite provides immediately accessible tools that requesters can use to set up writing tasks on their smart watches [112]. Collaborative platforms were introduced, allowing teams of expert crowd workers to manage a project with complex tasks and dynamically organize their team [132, 133]. The Huddler system assembles a stable team of familiar crowd workers to facilitate collaboration on complex tasks [134]. Salehi et al. explored various communication mechanisms to study how workers can communicate asynchronously with the requester for context transfer in writing tasks [135].

2.3.3 Crowdsourced Music Performances Using Smartphones

There is a definite separation between audiences and performers in a traditional musical concert. However, musicians and composers often try to blur this line by making audiences involved in the performance. This happens in a wide variety of genres from experimental to popular music. In Jean Hasse's composition *Moths* [136], audience was instructed to whistle along to a conductor's gestures and a graphical score. The role of the audience in this piece was to play music as a performer, and the whistling of the audience was the only sound of the performance. In popular music, audiences also often participate by making sounds directly such as singing, clapping, or stomping feet (e.g. We will rock you by Queen). In this section, we limit our scope to audience participation in music performance that uses technologies, specially after the emergence of smartphone as a medium for the participation. The long standing history of audience participation in music is available in [137].

Many have suggested that Dialtones (A Telesymphony) by Golan Levin is the first work that incorporated large-scale audience involvement using mobile phones in a music performance [138]. Although Dialtones(A Telesymphony) is the first use of audience mobile phones in large-scale music performance, researchers who have studied audience participation note that it is not audience participation due to their passive roles [19, 139]. McAllister et al. utilized PDAs to capture and transmit the graphic gestures of participants, sampled from the audience, for a music improvisation [139]. Net_Dérive by Tanaka was an audiovisual installation in a gallery where the location of three participants were deployed as materials for visualization and sonification of the installation [140]. The Stanford Mobile Phone Orchestra held their annual concert with the theme of audience participation [141]. In TweetDreams, Twitter tweets containing certain hashtags were sonified and visualized with their textual content so audiences could trigger audiovisual events in real time at the performance venue [142].

The Web Audio API [143] accelerated emerging trends of web browser-based music applications where sound is synthesized and generated directly from web pages. While the web-based application will approach the level of native audio applications in the near future, for example building DAW on the web browser [144], its nature of accessibility and connectivity invites collaborative music making systems. Especially, the web audio API that runs on commodity smartphones attracts musicians to distribute interactive music applications and to democratize music by moving sound sources from speakers on stage to smartphones on people's palms. The majority of music performances presented in the first Web Audio Conference involves audience participation (or audience involvement)[145, 146, 147, 148, 149] and this shows the strong focus of collaborative music making with the audience in the web audio community. Such web-based audience participation requires less

configuration for each individual compared to other methods. The application should run simply by launching a web page with a given link. The app should run regardless of the platform and the device type as long as a modern web browser with Web Audio runs on the device. Lastly, this differs from the previous approaches where audience influences music indirectly and sound is coming from the stage.

2.3.4 Observational Studies in Real-time Collaboration

This work draws ideas from and extends the long tradition of studying collaborative design and real-time groupware in the design context. Observational methods in the collaborative design context are well exemplified by [150]. The findings from such studies have been used to generate design implications of current collaborative systems. A comprehensive summary of research on awareness and coordination in collaborative systems has been done by Gutwin and Greenberg [151]. Our methodological approach resembles that of observational studies where distributed workers verbally communicate with some shared visual context [152].

CHAPTER 3

A User Study on Real-Time Collaboration in a Crowd-powered Sketching Tool²

3.1 Introduction

In recent years, interactive crowdsourcing systems have been designed for end users to send requests directly to crowd workers, and to continuously collaborate with them in real time [153, 154]. In such systems, requesters can verbally communicate with workers to request various tasks — ranging from accessibility assistance to writing software — and researchers have developed a number of crowd-powered systems to enable this novel collaborative model [119, 63, 118, 127, 112, 155, 60]. This form of direct and natural interaction between a requester and crowd workers in real time minimizes the efforts needed in traditional crowdsourcing workflows, where a task must be broken down into context-free microtasks and the responses aggregated afterward [110]. Real-time collaboration between requesters and workers is useful in expanding the application of crowdsourcing to broader domains, allowing workers to solve open-ended tasks whose steps are unknown in advance and may change during the process.

While the mode of end users directly collaborating with crowd workers unlocks new opportunities for crowdsourcing, end users being requesters poses challenges beyond mere unfamiliarity to such collaboration. This is because the characteristics unique to crowdsourcing — ubiquity, scale, transiency, and anonymity — make using crowd-powered systems different from simply collaborating with other users via real-time groupware. Specifically, designing interactive crowdsourcing systems entails an inherently asymmetrical structure between a requester and crowd workers in various ways: scale, roles, expertise, communication channels, and user interfaces. Such differences and imbalances are embedded in interactive crowd-powered systems and pose new challenges beyond those addressed in

²Portions of this chapter appear in [56]

real-time groupware design [156]. However, the current body of research in real-time crowd-powered systems has focused on developing interactive crowdsourcing systems for a specific domain, in which the system that enables workers to resolve certain types of tasks is the main contribution [119, 63, 118, 127, 112, 155, 60]. Furthermore, the validation process of such systems is often overly simplified, having just one requester throughout all study trials to control for variance among requesters [129, 155, 60]. We lack sufficient understanding of the challenges that requesters have as end users of interactive crowdsourcing systems in general. This work focuses on examining the requester side of crowdsourcing. The findings from this work can contribute to improving the existing systems and informing the design of future interactive crowdsourcing and mixed-initiative systems.

The goal of this work is to better understand challenges present when a requester collaborates and verbally communicates with crowd workers in real-time collaborative crowdsourcing systems, and to formulate design recommendations from the findings for future interactive crowdsourcing systems.

The inherently transient nature of crowdsourcing poses a set of challenges that are not present in previously existing real-time groupware. In addition, designing a single system for two disparate groups — requesters and workers — may involve different approaches for each group, which can create unforeseen challenges. In particular, we are initially motivated by the following questions: Q1) How would requesters find and understand the benefits of real-time collaboration in crowdsourcing and engage with workers? (3.4.1) Q2) How would the potential asymmetry of expertise, roles, or communication channels between a requester and crowd workers impact their communication and collaboration?(3.4.2,3.4.3) Q3) What particular challenges do requesters and workers face in collaboration?(3.4.4,3.4.5) Overarching all these questions is a desire to discover how can we address these issues in the next iteration of the system and inform the design of future crowdsourcing systems.(3.5)

To answer these questions, we conducted a user study of investigating how requesters collaborate and communicate with workers via an interactive crowdsourcing system. We chose to use an existing crowd-powered system, SketchExpress, which allows requesters to create sketch prototypes of graphical user interfaces with crowd workers [155, 60]. In the study, an end user (requester) is asked to verbally describe and draw a GUI sketch; crowd workers behind the scene listen to the description and view the sketch, and create a refined GUI in real time. While the authors of previous works focused on validating systems for crowd workers to effectively create digital artifacts, in this study we focus on how various requesters use the system and collaborate with workers [155, 60].

Our main contributions are findings valuable beyond the scope of crowd-powered design tools, derived from a qualitative analysis using questionnaires, interviews, and observations.

The system and task in this study involve common components of real-time collaboration in crowdsourcing: requesters, crowd workers, an artifact (sketch), and visual context (canvas) that are shared remotely, and communication channels (voice/chat). This structure can be found in other interactive crowdsourcing systems and helps us generalize our design recommendations. Following is a summary of our findings:

- Requesters actively collaborate with workers and understand the benefits of working with them in real time(3.4.1).
- Expert requesters may speak in a style that makes it difficult for crowd workers to comprehend their requests (less descriptive, more jargon)(3.4.2).
- The asymmetry in communication modalities (speech vs. text) causes confusion(3.4.3).
- Simply sharing the visual context was not enough to effectively coordinate collaboration.(3.4.4)
- Speech pace varies among individuals; rapid speech may result in a backlog of requests.(3.4.5)

The design recommendations drawn from the study contribute to the broader goal of enhancing the design of crowdsourcing systems that can facilitate effective real-time communication between requesters and crowd workers.

3.2 Apparition — System Description

In order to investigate the challenges that requesters face in communicating and collaborating with crowd workers, we conducted a user study using SketchExpress, a web-based sketch prototyping tool [155, 60]. In prior user studies for validating the systems [155, 60], one of the authors played the role of the requester to eliminate the variability that having multiple requesters would introduce, while crowd workers were recruited on the fly from Amazon’s MTurk platform [108]. While this study design is beneficial to confirm if a system is functionally valid, the effectiveness of systems with diverse end users in practice remains unexplored. We focus on the challenges that requesters with varying backgrounds have in using a crowd-powered design tool. In this section, we briefly introduce the features that are designed to support communication and collaboration.

In SketchExpress, a requester can naturally and continuously interact with crowd workers to create an early-stage graphical user interface prototype. When the session starts, a requester verbally describes the prototype that they want to create. At the same time, a

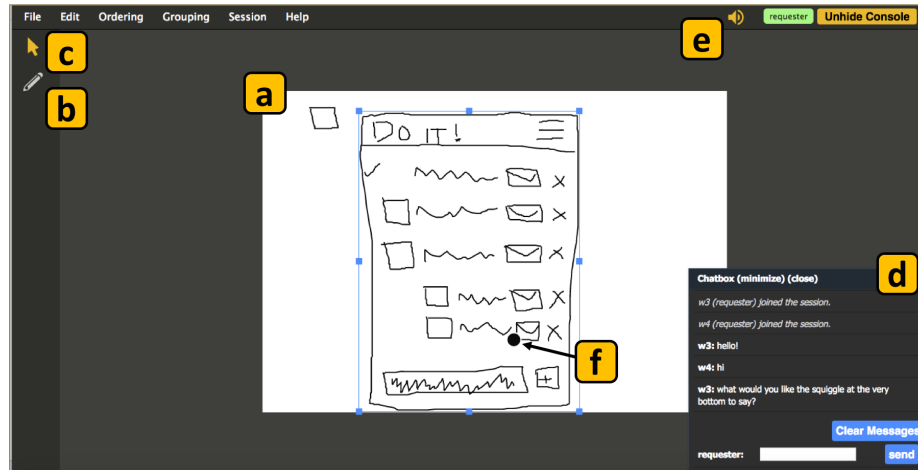



Figure 3.1: The requester interface of SketchExpress supplies a variety of tools that facilitate sketching and communication with crowd workers: (a) shared canvas, (b) pencil tool, (c) selection tool, (d) chat box, (e) mute/unmute button, (f) circles to visualize requester click events.

requester can draw a rough sketch of the GUI while narrating what they are drawing. The canvas is shared in real time, so crowd workers can see the requester’s in-progress sketch and hear their verbal description (see Fig. 3.1-a). State synchronization occurs at a per-element level—that is, an element appears on the other side once it is created, but not during its creation. Through the interactive tutorial, crowd workers are instructed to listen to the verbal description, inspect the rough sketch that the requester draws, and *replace* each sketched element with a proper shape provided by the system. Fig. 3.2 shows an example (T1) of what a requester can draw on a canvas, and demonstrates how crowd workers gradually convert a rough sketch (T2-a) into a more refined one (T2-c).

SketchExpress provides two different sets of tools for requesters and crowd workers. The requester is able to focus on verbally describing and roughly sketching the desired GUI prototype. They are provided with two tools for sketching: a *pencil tool* (Fig. 3.1-b), with which a user can draw free-form drawings; and a *selection tool* (Fig. 3.1-c), with which a requester can select any element on the canvas to transform (move, resize, rotate) or delete it. The simplicity of these tools helps keep the requester’s interaction with the system natural (free-form drawing and speech) so that a user with little experience using advanced computer graphics tools can still use them successfully. On the contrary, crowd workers have access to a larger number of functions than requesters have. They can add various types of GUI elements (e.g., buttons, arrows, text boxes), and change their properties (color, opacity, stroke width, and stroke pattern). These functions are similar to those available in commodity prototyping tools and slide-based presentation software (e.g., PowerPoint).

Therefore, crowd workers with some experience using such software can begin using these tools immediately. For more details on the worker interface, see [155, 60].

SketchExpress supports communication between a requester and crowd workers in two distinctive ways. First, it implements real-time audio streaming using WebRTC for a requester to directly speak to crowd workers [157]. The authors of SketchExpress wanted requesters to have the advantages of multi-modal interaction in simultaneously speaking and sketching to expedite the creation process, similar to collaboration with a co-located colleague [155]. However, audio streaming is one-sided in the case of SketchExpress, so crowd workers cannot speak to the requester. The authors of Apparition found that streaming audio from crowd workers can cause problems: the crowd’s voices may drown out the requester’s, or the conversation may digress quickly if there are multiple people talking to each other. A requester can mute the microphone (Fig. 3.1-e) to stop the audio stream as needed, or to avoid distracting workers. When muted, its icon changes to . The envisioned interaction of the original system resembles that of an intelligent system in which a requester can describe their interface idea, and their idea comes to life in a short span of time [155]. However, in practice, crowd workers often want to communicate with the requester to ask questions, to clarify the request, or to report technical glitches. To that end, SketchExpress includes a text-based message interface that resembles a typical instant messaging application (Fig. 3.1-d) [60]. A requester can view messages and respond to them either verbally or in text.

It is worth noting that this kind of asymmetry in communication modalities (text vs. speech) is not common in other groupware and social computing contexts. Similar setups can be found in both game streaming services (e.g., Twitch) or other live streaming services (Facebook Live, Periscope), which recently have been studied for the effects in fostering participation and engaging spectators [22, 27].

Lastly, the shared canvas in SketchExpress, which is synchronized in real time on both ends, not only allows both groups to collaborate on a single, final copy in real time, but also builds conversational grounding among workers and requesters [152]. In addition to the real-time, synchronized canvas, SketchExpress provides a way to visualize the requester’s mouse clicks; when a requester clicks a location on the canvas, it will generate a colored circle at that location which is shown to both the requester and crowd workers. The circle fades out after a few seconds (Fig. 3.1-f). The click visualization is useful when a requester wants to indicate a specific position on the canvas where a UI element should be created. For example, a requester can say “*I want a label that says ‘Name’ here (click!)*” without drawing the word ‘Name’ with the pen tool.

SketchExpress is one instance of interactive crowdsourcing systems that support real-

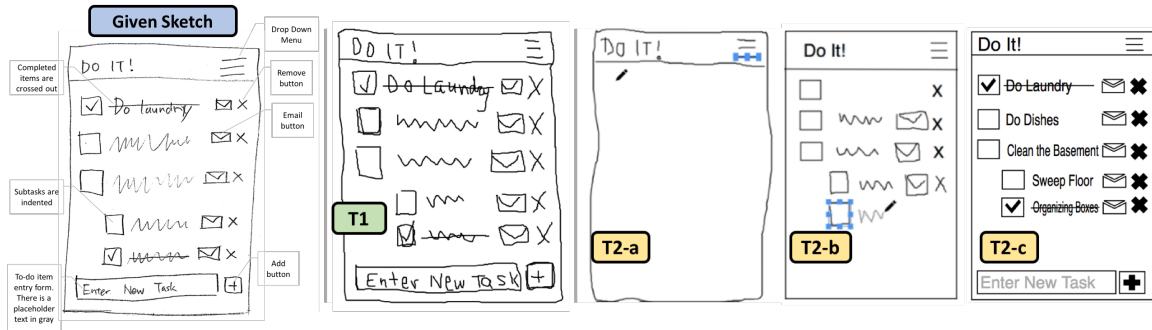


Figure 3.2: **(Given Sketch)** The sketch of a to-do list application given to participants (requesters). **(T1)** A sketch hand-drawn in SketchExpress in Task 1 without crowd workers. **(T2)** Progression of a sketch in SketchExpress in Task 2 as T2-a) it is first hand-drawn by the requester (same drawer as in T1), T2-b) the requester continues hand-drawing and crowd workers begin replacing hand-drawn pieces with higher-fidelity shapes and icons, and T2-c) the crowd workers replace all hand-drawn pieces.

time collaboration. However, the structure of the system can be generalized to other similar systems, particularly a requester and crowd workers collaborate to create a digital artifact in real time. The structure includes: 1) one end user (requester) who leads the communication with concrete goals, 2) a set of crowd workers, and 3) an interactive system for collaboration and communication, the components of which are 3-1) a shared (visual) artifact, 3-2) communication channel (audio streaming and chat), and 3-3) self-coordination tools for workers. The findings from the study will be applicable to the crowdsourcing systems that have the same structure in various applications. In the following section, we describe the method of studying challenges in communication.

3.3 User Study

The goal of this work is to better understand the way in which requesters communicate and collaborate with workers. We conducted a user study with SketchExpress and run a qualitative analysis, examining multiple aspects of the communication between requesters and crowd workers in order to identify the challenges and common patterns that people exhibit during collaboration. Our method draws ideas from observational studies in collaborative design [158, 150]. In particular the setup of SketchExpress is similar to the case where distributed workers communicate with shared visual context [152]. Here we introduce detailed study procedure, participants and our limitations.

3.3.1 Study Procedure

We conducted the study with 10 participants recruited locally and 20 crowd workers recruited from Amazon Mechanical Turk, an online crowdsourcing platform. The 10 participants came to the lab and completed tasks using SketchExpress as requesters. First, requesters completed a brief demographics survey and were given instructions on how to use SketchExpress. Requesters were then asked to complete two tasks. The order in which the tasks were completed was counterbalanced. In both tasks, requesters were shown a low-fidelity prototype (Fig. 3.2-Given Sketch), which was a sketch of a to-do list application for a smartphone. We chose the to-do list application primarily for its familiarity with a broad population. We presented a low-fidelity prototype to the requesters (not workers) because they were asked to convey the general idea of the UI without needing to meet small details. We included annotations in our example GUI in order to keep participants from asking the study moderators clarification questions directly, as their explanations could bias how requesters verbally described the GUI.

Here are the descriptions of the two tasks given to the participants during our lab studies: **Task 1 (T1)- Hypothetical Asynchronous Scenario (no Crowd Workers):** *Requesters were shown the example GUI and were asked to draw and verbally explain the interface in SketchExpress as if they were giving a task to a colleague who would be creating a sketch prototype of the GUI. Requesters were told the goal of T1 was to convey the sketch idea and to help the hypothetical colleague create a polished prototype of the example GUI.*

Task 2 (T2)- Real-time Collaboration with Crowd Workers: *Requesters were shown the example GUI sketch and were asked to draw and verbally explain the GUI to crowd workers who would be creating a sketch prototype of the GUI in real time in SketchExpress. Requesters were told the goal of T2 was to create a polished prototype with the crowd workers.*

For both tasks, we asked the participants to stop working on the task when they were satisfied with the outcome; we stopped task execution ourselves only if it took over half an hour. The study moderators then conducted a follow-up interview to gain further insights into the requesters' experiences. We asked requesters questions about their experience communicating and collaborating with crowd workers, as well as their experience using SketchExpress. In all sessions, we recruited two workers, with the exception of P8's session, during which one of the two workers left mid-session.

The work primarily focuses on the qualitative analysis of T2, the task for which a requester directly collaborated with the crowd workers. Having a hypothetical scenario of a user making a request asynchronously (T1) only gave us a reference point to better understand the communication styles used in T2 (such as how sparse a user's speech is in

T2 and how requesters choose to communicate in response to workers making changes in real time). In addition, we wanted to have the participants experience making a request in an asynchronous fashion and help them to reflect on what it would be like to work in two different scenarios when they were asked relevant questions during the follow-up interview. As we were not interested in comparing the performance (time taken and quality of the artifact) of crowd workers in two distinct approaches (synchronous vs. asynchronous), we did not need to have T1 completed by crowd workers separately.

3.3.2 Participants

3.3.2.1 Requesters

Ten requesters were recruited to complete the user study. Five were designers (denoted by P1, P2, P3, P4, and P5) and five were non-designers (denoted by P6, P7, P8, P9, and P10). Four designers and one non-designer were female. The average age of the designers was 30 (ranging from 27-37) while the average age of the non-designers was 24 (ranging from 19-38). None of the non-designers had previous experience designing a GUI, while all the designers had prior experience in various fields, including UI, UX, product design, and graphic design. The designers varied greatly in experience, ranging from 1 to 12 years of experience. Each participant was compensated for their time with \$15 in cash after the session.

3.3.2.2 Crowd Workers

We recruited 20 unique crowd workers from Mechanical Turk who had not previously used SketchExpress, and who were all U.S.-based and had an approval rating higher than 80%. Of the 20 workers, 19 completed a demographic survey. Their average age was 32, and 8 of the 19 crowd workers were female. When asked to provide a rating of their design expertise on a five-point Likert scale, the crowd workers' average score was 1.72 ($\sigma = 1.07$), with 1 representing having little design experience and 5 representing being an experienced professional designer.

The crowd workers were recruited at the beginning of T2. The moderator of the study posted a task in MTurk using LegionTools [121]. The task had four binary questions to assess the eligibility of potential participants. If the worker was eligible for the study, they were asked to watch a tutorial video and were then routed to the task page. Once there were enough crowd workers in a session, typically two workers, the moderator checked if workers knew how to use the basic functions of SketchExpress and trained them by going over a sample task that simulates the main task (T2). Once the workers completed the sample task,

the participant started the main task (T2). After the task, workers were asked to fill out a survey and got paid at the rate of \$0.17 per minute (\$10.20 per hour).

3.3.2.3 Knowledge of Background

Note that neither requesters nor crowd workers were explicitly aware of each others' expertise or demographics. Workers did not know whether or not a requester was a designer, and requesters did not know crowd workers' backgrounds. The study moderators did use the term "crowd workers" when providing the requesters study instructions, but did not provide any further details about them, as we did not want to influence requesters' communication approaches. This accurately reflects the temporary nature of current practice in crowdsourcing, where requesters do not know who crowd workers are, necessitating that systems themselves mitigate the resulting uncertainty.

3.3.3 Data Analysis

In order to capture the complete speech and canvas activity in SketchExpress, the audio and computer screens of T1, T2, and the follow-up interview were recorded. Both the audio extracted from the videos and the interviews were transcribed for data analysis. The transcribed speech from T1 and T2 was segmented into a series of speech utterances with a starting timestamp. The boundaries of each speech segment were drawn at pauses, at the end of sentences, or at a change in topic. Conversation coding has been used in previous studies that investigated shared visual contexts in collaborative design [159, 160, 161]. The coding labels we developed were inspired by [161], which introduced the following labels: *indicative*, *demonstrative*, and *descriptive*. Then, we developed and refined the coding labels and interview questions over the course of pilot studies (eight runs) in the context of the initial research questions that we had (listed in the Introduction). As we observed the study, conducted interviews, transcribed the recordings, and coded the conversations, we sought to identify emerging themes in relation to expertise, engagement, and temporal patterns. One of the authors coded all participant trials, and these coding results are what we present and analyze in sections 5 and 6. We also had another author code 10% of the data to report the inter-rater reliability of our coding. We calculated intraclass correlation coefficients (ICC) for the two coders' coding, and we report these scores and their interpretation (*poor*, *fair*, *good*, *excellent*) according to [162] following each of the label definitions below.

- ***Descriptive***: The requester describes how an element should look or describes the element's shape and geometrical properties (e.g., circle, rectangle, lines). (ICC : .746, *good*).

- **Verbally Referential:** The segment contains words (including 'deictic expression') or phrases that refer to specific elements or a specific area of the canvas (e.g., "here", "the left side of the screen"). (*ICC : .717, good*).
- **Diectic Gestures:** The requester uses the mouse cursor to point to a certain area or element on the canvas during the speech segment. (*ICC : .834, excellent*).
- **Click Events:** The number of click events observed during the segment; during the task, click events were visualized with small circles on participants' screens (Fig. 3.1-f). (*ICC : .796, excellent*).
- **Corrective:** The requester asks workers to modify existing elements they have created. (*ICC : .547, fair*).
- **Repetitive:** The requester repeats a request that has been made earlier in the session. (*ICC : .788, excellent*).
- **Feedback:** The requester gives positive/negative feedback to workers (e.g., "That looks good, thank you."). (*ICC : .830, excellent*).

Note that these labels are not necessarily mutually exclusive; a given segment can have more than one label. We also reviewed the chat history between crowd workers and the requester, which is interleaved with the audio recording of the requester.

3.3.4 Limitations

(In-lab Study) One of the current limitations in this study may come from the artificial setting in which a hypothetical task was provided to the participants by the authors. Instead, we opted for a task that balances flexibility and consistency, as having a controlled task precludes communication being affected by the varying difficulty of creating a given artifact. A task was selected to simulate a hypothetical scenario in which participants have a conceptual idea in their mind. In general, we find that the task selected was a reasonable compromise for exploring the communication and collaboration process without having to implement the full pipeline of the system in the wild for a longitudinal study.

(Lack of Discussion on Performance) We are limited in relating the findings with how effectively and efficiently the requester could produce the sketch-prototype as we focus on only the requesters' side of the process. Although we evaluate the performance of crowd workers (quality and time), the outcome can be a compound outcome of many different factors, not only the ones that we are interested in for this study (i.e., a requester's

communication with workers), but also workers' abilities (e.g., the skill level of workers in using the tool, communication and collaboration among workers), which are beyond the scope of this work. In addition, the expertise of crowd workers was not considered nor strictly controlled.

(Limited Number of Sessions and Workers) Lastly, it is noteworthy to mention that the number of sessions ($n = 10$) is not enough to quantitatively verify the findings. Rather, our study is primarily qualitative. While we cannot make comparative claims that require statistical significance (e.g., designers vs. non-designers, or T1 vs. T2), the qualitative analysis provides us with a thorough understanding of the observed behaviors through interviews, conversation coding, and video review. Discovering problematic incidents can be done with a small number of participants and can clearly inform how we can improve the current design of the system and similar future systems [163]. It is not uncommon to have a small scale in such studies, especially for collaborative settings that involve more than one participant—for example, three to ten sessions [164, 165, 127]. In addition, the number of crowd workers was typically four to five per session in the original system. We kept a smaller number of workers to simplify the factors coming from collaboration between workers (e.g., worker-to-worker communication, self-coordination).

3.4 Result

We recorded completion times for both tasks per participant. The mean completion time for Task 1 (T1, without crowd workers) was 4.9 mins ($median = 4.5, \sigma = 2.2$). The mean completion time for Task 2 (T2, with crowd workers) was 19.7 mins ($median = 20, \sigma = 6.7$). Note that completion times for the two tasks are not comparable, because T2 included an extra stage of completing the task with crowd workers and T1 did not. We also evaluated the final crowd-created sketches in T2 against a rubric to understand the effectiveness of the requester and crowd collaboration. The rubric measures recall of UI characteristics of the original sketch we provided to requesters. Across the ten T2 trials, the mean recall was 86% ($median = 94%, \sigma = 15%$). This demonstrates that a requester and crowd workers can effectively collaborate to create a medium-fidelity prototype in real-time. The reported recall score can be considered effective, given that we did not provide requesters an explicit list of requirements alongside the sketch.

Based on the observations that we made from the transcriptions and video recordings, we present our findings in five different themes that emerged from the results and are relevant to the three motivational questions we presented in the introduction.

3.4.1 Real-time Collaboration with Workers in an Interactive Crowdsourcing System

In this section, we briefly discuss how the verbal communication patterns in Task 1 (T1, without crowd workers) and Task 2 (T2, with crowd workers) differ. In addition, we would like to understand why and to what degree participants would or would not prefer this interactive crowdsourcing system over potential asynchronous collaboration.

3.4.1.1 Requesters' Involvement in Real-time Collaboration

We observed that requesters' speech and drawing activity in T2 is more sparse than in T1. The average number of words per minute (WPM) in T2 is lower than it is in T1 ($T1: 75 \text{ WPM}$ ($\sigma = 25.4$), $T2: 39$ ($\sigma = 15.6$))¹. This is not surprising, given that the requester, during T2, was often silently monitoring the progress of the sketch and checking the elements that crowd workers created. Requesters then interrupted and gave feedback (either positive or negative) to crowd workers in response to the progress that they made (avg. 13.2 times per session). For instance, *"I like the spacing in do laundry between the check box"* (P2), or *"no, no, no you can either get rid of these or put them over here"* (P9). In addition, requesters made corrective requests for crowd workers to modify existing elements (avg. 10.6 times per session). For example, the second comment by P9 above is a corrective comment. As such requests are typically made after the requester sees the actual elements that crowd workers have created, the corrections made with these comments would have resulted in incorrect elements if the crowd workers had worked asynchronously, as in T1. We looked in further detail at the corrective requests made in P9's T2, comparing them with the corresponding requests in their T1, and found out the corrective requests that P9 made were not simply due to the mistakes that workers made. Rather, we found that the majority of the corrective comments (76.7%) were 1) something that was under-specified in T1 (16.8%), 2) something that was visually implied but not explicitly mentioned in T1 (e.g., alignment, position) (38.3%), or 3) made on the fly during (21.5%); the higher fidelity of the actual outcome made the requesters specify a new element property. Examples for such comments from participants in each category are as follows:

(P2) *"If this text can be moved over to left aligned that would be great."*

(P1) *"Can you make this narrower? This is a mobile application but the screen looks like a desktop web page not mobile."*

¹Both numbers are lower than the average (150 WPM) for normal conversation [166].

(P9) *“Let’s try to do a green check mark. (choosing the color of the check mark)”*

This indicates that the request made in T1 is not necessarily sufficient for crowd workers to produce exactly correct outcomes. Consistent with findings in collaborative design literature, the process of co-creation provides richer information to the participants than the resulting artifact [150] This clearly displays the benefits of real-time collaboration in crowd sourcing, at the price of more involvement in completing the task.

3.4.1.2 Participants’ Reflection on Real-time Collaboration with Workers

In the follow-up interview, the participants were able to understand the benefits and challenges of real-time collaboration with workers in comparison to the asynchronous collaboration scenario. We asked the requesters which approach (T1: asynchronous vs. T2: real-time crowdsourcing) they would use in practice. 7 of 10 requesters preferred real-time collaboration over asynchronous collaboration. Requesters found *“benefits in being able to give feedback in real-time since the corrections could be made as they go rather than after”* (P6). P4’s comment on this question highlights the benefits of real-time collaboration well:

(P4) *“It’s much less painstaking for me to give them live criticism than to make sure I talk out every detail in some sort of instructional video that I send off.”*

Three requesters (P2, P3, P5), all of whom are designers, indicated a preference to work asynchronously as they were in favor of being able to work in parallel. They expressed their frustration with watching the workers’ creation process. At the same time, they also understood why they, the requesters, might want to work in real-time, reflecting their prior experiences in working asynchronously. P3 noted that working asynchronously is often followed by *“a lot of revision”* through multiple rounds of having to send a request, waiting for the task to be done, and then repeating this process, going back and forth until they are satisfied with the final result. The other two designers also expressed mixed feelings on their decisions.

(P2) *“Because while I was watching, I did notice that I was feeling a little bit stressed when they got it wrong. But I also know that what happens sometimes is you just get it asynchronously, and that’s also stressful, and that might even take more time, like explaining to them [workers] through email what’s wrong, what needs to be corrected.”*

As illustrated by these quotes, the designers even exhibited frustration while watching workers working in real time. They might have underestimated the difficulties that non-expert crowd workers would face [167], especially given that the designers could have

finished the prototype more quickly than the workers. A participant mentioned the potential of monitoring progress intermittently instead of in real time, and how such an approach might address their concerns on the continuous involvement with access to real-time progress on the work.

(P2) *“I think I like being able to see the progress, but I would want to be able to multitask. I would like to have this somewhere playing as I’m doing my own work, and then I glance up at it and see it.”*

This suggests a hybrid method of both synchronous and asynchronous real-time interaction, and this mixed approach can lead to productivity gains, as two groups can work in parallel with less interruption [63]. In general, the majority of the participants preferred such interactive crowdsourcing systems, even though it could have been an unfamiliar experience of working with anonymous remote crowd workers.

3.4.2 Expertise, Language, and Jargon

Reviewing the specific language that requesters used to describe their sketches, it appears that the level of expertise determines the ways in which participants describe GUI elements. As the designers are familiar with a certain set of UI design terms, they speak differently from how non-designers describe. For example, see the following examples of descriptions for the container window of the UI.

(P7) *“So I am gonna have like a box here which is going to be an interface like a rectangular box, its height is bigger than the width.”*

(P3) *“So to start, I am going to go ahead and draw the mobile device screen.”*

As seen in the example, the non-designer (P7) characterized the element by its shape and geometric properties. In contrast, P3 did not explicitly mention the shape of the element in their description.

In addition, non-designers tended to refer to elements in the canvas, using referential words or gestures with their mouse cursor. The participant (P7) in the example tried to provide visual context for their description, using deictic words such as “this” or “here”, and associating the drawing with speech. On the other hand, the designer (P3) did not refer to the element on the canvas even though the participant was drawing the container on the canvas. This may be because designers have a clear connection between the spoken terms and the visuals on the canvas in their mental model. However, the crowd workers may not have the same clear connection, which could lead to a delay in comprehending the request [131]. To

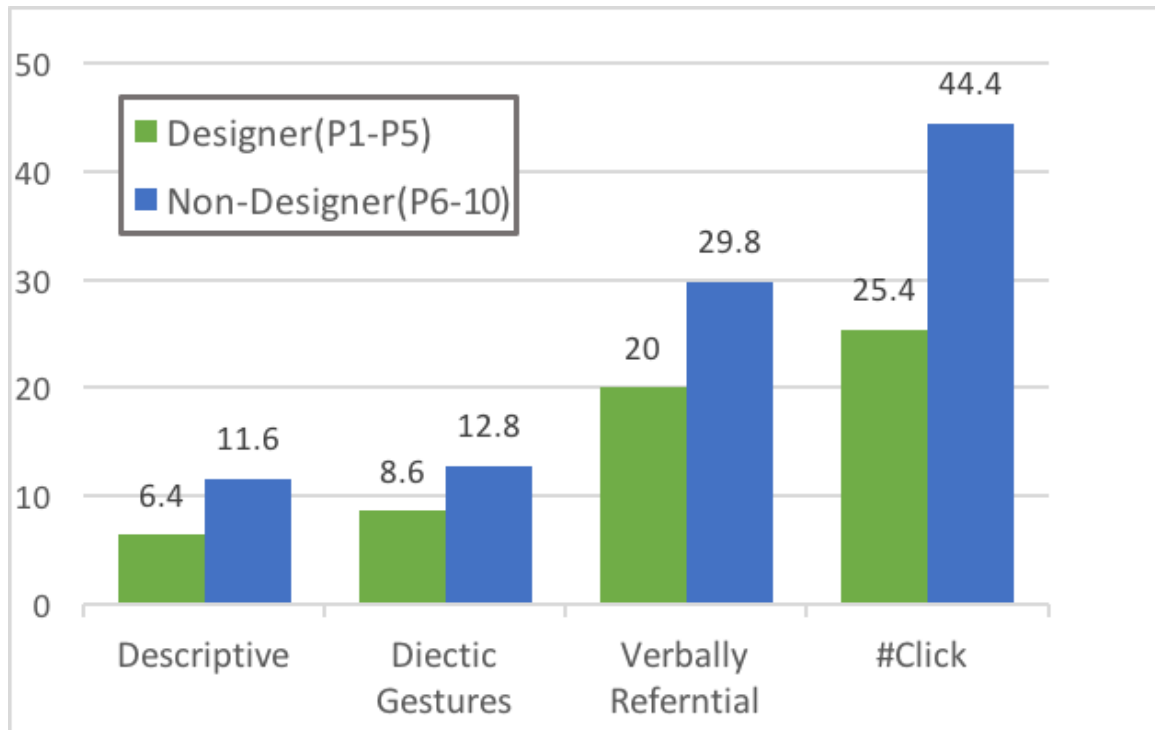


Figure 3.3: On average, the non-designers described the request in more descriptive ways and referred to elements more frequently when working with workers

see if this communication style is consistent across participants, we counted the number of descriptions that are *descriptive* and *verbally referential*. Furthermore, we annotated the videos to see if they contained elements of *referring gestures* (e.g., a gesture of moving the mouse cursor in a small circle on top of an element) and *clicked* elements to generate click visualizations (Fig. 3.1-6). On average, the frequencies of descriptive expressions and reference to elements during the tasks were greater for non-designers than designers (Fig. 3.3). Similarly, non-designers tended to refer to elements with the mouse cursor (click or gesture) more than designers (Fig. 3.3). A previous study showed that having a shared workspace encourages users to rely on visual information to provide the necessary communicative and coordinating cues [168]. In our study, requesters without expertise seem to have the stronger tendency of using visual cues whereas requesters with expertise showed less reliance on them.

From a crowd worker’s perspective, the same kind of unfamiliarity with UI design terminology might have been an obstacle in comprehending a designer’s requests quickly, locating elements on the canvas, and catching implicit requirements. For example, when requesters asked workers to create the outer box of the application, the implicit requirement that the mobile phone screen be in “portrait” mode was missed by the workers in four out of

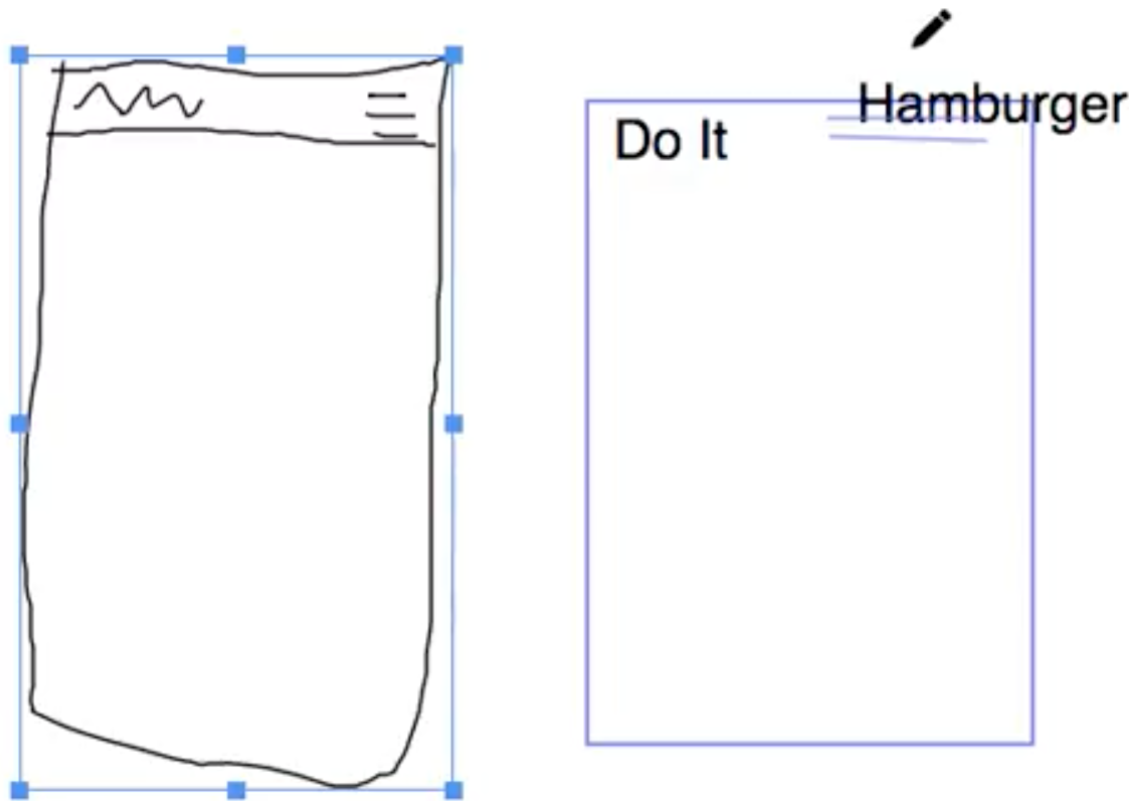



Figure 3.4: *Hamburger menu gone wrong*: P1 used the jargon “hamburger menu” and a crowd worker spelled it out, as they were not familiar with the term.

ten sessions, and the requesters needed to correct them later. Another interesting example occurred when participants described the menu button, which is labeled as “drop down menu” in Fig. 3.2-(Given Sketch). The following are two examples from P8 and P1.

(P8) *“I want to create like a drop down menu in the upper right-hand corner with like three little lines.”*

(P1) *“I want you to add a hamburger menu on the top right corner.”*

In this quote, a designer (P1) and a non-designer (P8) described the same element in two distinct ways. Overall, three out of five designers (six out of eight if we include designers from the pilot studies) used the jargon “hamburger menu”, which refers to . Zero non-designers used this jargon, and four out of five non-designers described the button based on its shape, with the phrase “three (parallel) lines”. Using such jargon could lead to confusion if crowd workers are not familiar with a term and designers do not clarify it further. For example, a crowd worker in designer P1’s session actually created a label that reads “hamburger” (Fig. 3.4.). This example illustrates the difficulties that non-expert crowd workers may have in comprehending an expert designer’s descriptions. When there exists an expertise gap in dyadic communication, the process of assessing, supplying, and acquiring expertise is required to reach common ground [169]. However, the temporary and task-oriented nature of crowdsourcing seems not to foster this communication, and as a result, expert requesters may fail to realize that crowd workers sometimes have trouble comprehending their requests.

Arguably, this challenge can be mitigated if the system recruits expert crowd workers with design backgrounds from other crowdsourcing platforms, such as Upwork. However, if we recruit workers with expertise, the main benefits of crowdsourcing are likely to fade away; such workers may not be as cost-efficient, scalable, and available as a general online crowd. The current body of crowdsourcing research tends to keep the benefits and to close the gap between expert support and crowdsourcing by developing computational tools that mediate the crowd’s efforts [155, 119, 60, 111, 112].

3.4.3 Speech, Text, Asymmetric Communication

As mentioned previously, using speech is essential to support multimodal interaction for a requester (speaking while drawing) [170]. The authors of SketchExpress chose to support multimodal inputs for the requester due to a set of advantages in expediting the creation process and capturing multiple information in a sketch concurrently. Indeed, the requesters in the study frequently performed mouse interactions (i.e., click, gesture, drawing elements,

drawing to indicate) while they were speaking (48.4% of the total speech segments). In addition, requiring crowd workers to type responses instead of speaking them is necessary to allow requesters to make their verbal requests clearly, without interruption (e.g., A requester’s voice can be easily lost in multiple crowd workers speaking simultaneously). Consequently, one of the unique characteristics of SketchExpress is the asymmetry of communication modalities: a requester speaks to workers and they respond by typing or making changes on the canvas. We discuss the challenges that arise due to asymmetric communication between the groups.

3.4.3.1 Lack of Immediate Responses

When we asked requesters *if they were bothered that they could speak to the workers while the workers could not speak to them*, 7 out of 10 of requesters were not in favor of asymmetric communication. Three of them were not in favor of the inequality because they simply found it awkward and unnatural to have different modalities (P6, P8, P5). Five of the participants expressed frustration at the lack of immediate responses from the workers when requests were made (P10, P2, P3, P4, P5). Conversing with the system without immediate responses may have made the communication feel one-sided, which can account for the reason why some participants thought it was awkward.

In general, network delays in groupware discourage participants from collaborating with each other [171]. However, the challenge in SketchExpress seems to come from the asymmetric communication. When crowd workers respond in the chat box in text, the requesters often missed the textual confirmation from the workers. Another study that has a similar setup also addressed the problem of participants missing textual comments from the workers [127]. Reversely, it is likely that workers’ textual messages may not receive a follow-up verbal response from the requester or other audience members [172]. On the other hand, lots of times there was no response at all; crowd workers started working without confirming their receipt of the request. However, starting to work on the request may not immediately provide any visual evidence that the request is being carried out through the synchronized canvas (e.g., navigating menu items) [156].

The absence of affirmative responses from the workers and the delay in creating elements may reduce overall workspace awareness in the system. Indeed, three requesters (P4, P7, P10) asked the moderator during the study if the workers were still there, having observed no activity both in the canvas and the chat box. P4’s suggestion on having simple visual responses from the workers exemplifies their concern with the system:

(P4) “...you kind of wait to see ‘Are they working on it? Did they not get it?’

Should I be typing this out?’. Yeah, it’s too many what-if scenarios in my head. Like if there could even be a button that says ‘Yep, got it’, so they could say ‘Cool, got it’ in a single click, then I would feel better knowing that they’ve understood what I just said.”

There were other relevant incidents that demonstrate the potential loss of having no responses. During T2 sessions, three requesters accidentally forgot to unmute the microphone when they were making requests. Due to the one-sided communication, it took a while for the participants to find out something was wrong and the mic was muted. For example, P9, who had muted the mic for 4.5 minutes, did not know for how long the crowd workers missed his verbal request(s). The participant expressed their difficulty in having awareness when asked *if verbal communication was effective* during the interview.

(P9) “...I used filler words. Just as I didn’t know that they were there, I kind of wanted them to know that I was there, even when I wasn’t talking.”

The participant verbally created an indication of presence for the crowd workers. This suggests the need to reinforce workspace awareness, as otherwise, unnecessary utterances may be used to remedy the perceived ignorance of the other group [173].

3.4.3.2 Mixed Ephemerality Creates Confusion

We observed that the mixed ephemerality of text and speech causes confusion in communication. In SketchExpress, chat messages are archived in the chat box so that both workers and requesters can browse the chat history. However, spoken words are ephemeral, leaving no record of having been uttered. Immediately, it was a challenge for crowd workers to recall all the spoken requests made in the past (Fig. 3.5-3). To reconcile this challenge, some requesters (P2, P8) voluntarily chose to use the chat box, as typing the instructions leaves a record, so that the crowd workers did not need to remember all the request details. Otherwise, when requesters did not type their requests, it was difficult to understand and recover the context from the chat history, as it only present one side of the communication. For example, Fig. 3.5 reflects 9 minutes of conversation in P7’s T2 session, in which a worker asked questions and responded to the requester. However, parsing the chat history in retrospect provides almost no additional information, as the spoken part are missing.

Furthermore, the archived textual history could add confusion, as it was not interleaved with the spoken instructions. For example, four consecutive “ok” or “yes” messages like in Fig. 3.5, can confuse a requester who may not know if the “yes” on the bottom was a response to their most recent request or an earlier one. When we reviewed the video


```
worker:Can I delete the scribble?  
worker:Ok  
worker:And what was next? :D  
worker:Ok  
worker:Ok.  
worker:Ok.  
worker:Ok  
worker:What was the next instructions?
```

Figure 3.5: 9 min-long history of chat during P7’s session. 1) Browsing the chat history does not help recover the context. 2) The worker asks for permission to delete the requester’s drawings (line 1). 3) The worker cannot recall the request (line 3 and 8).

recordings of the study sessions, it was difficult to discern if the message was from a worker who just responded to a particular request or if it was from the past. The requesters should have had the similar challenges. The chat box had a common design signifier to draw a user’s attention to a new message (the title bar turns yellow when a new message is received). However, as the requester can verbally respond, the highlight hardly ever went away, and the chat box generally remained constantly yellow.

Questions that remain unanswered in the chat box also create unnecessary interaction. Three requesters (P2, P5, P7) verbally answered a question and chose to later leave a textual answer in the chat box. One of them (P5) provided the textual answer for a worker’s question 70 seconds after she had verbally answered the question. The requester might have forgotten that she already gave an answer, or did not want to leave unanswered questions in the chat history. In any case, this created extra overhead for the requester physically and cognitively. From that point, she chose to use the chat box only and muted the microphone.

While we did not expect this asymmetric communication to be of interest in the study, we found a number of problematic incidents coming from this inequality. Live streaming services (e.g., Twitch, Facebook Live) also include asymmetric communication or cross-modal interaction (spoken and written), and yet are lacking analytic approaches [172]. We believe that addressing these challenges in general will help improve interactive crowdsourcing systems and other relevant live media.

3.4.4 Challenges in Co-working on a Shared Artifact

Having a shared visual context not only enables remote co-creation of the digital artifact, but also enriches the communication and collaboration process [168]. One unique challenge to an interactive crowdsourcing system is that the purpose of using the shared resource for

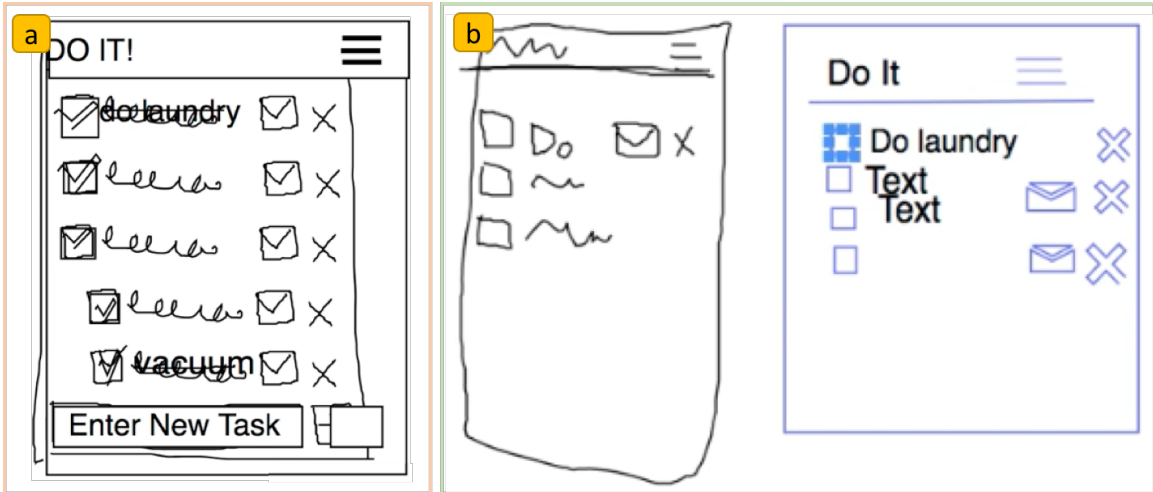


Figure 3.6: a) *Crowded canvas*: The workers place their higher-fidelity square checkboxes and task label text directly on top of the corresponding elements hand-drawn by the requester, instead of deleting the hand-drawn elements first. b) *Split canvas*: The crowd workers choose to preserve the requester’s drawing and create their prototype next to it.

requesters can be drastically different from the process for workers. Specifically, in this case, the rough sketch that a requester makes serves as a visual specification of the request (with corresponding narration) and the workers’ drawing is the final outcome of the system.

During the process of converting a hand-drawn sketch into a prototype, we observed that workers often hesitated to replace pieces of the requester’s drawing, even though during the training they were instructed to do so. In general, people naturally follow social protocols for mediating interactions, so it would have been unnatural for crowd workers to delete element drawings created by the requesters, even though they were instructed to do so in the interactive tutorial prior to the session [174]. Consequently, having two sets of the same element in two different fatalities (one hand-drawn by a requester, and the other UI element added by a crowd worker) can make the shared canvas cluttered quickly. We frequently observed cases where hand-drawn elements and their higher fidelity counterparts appeared overlaid, causing the canvas to be cluttered. Such cluttered elements not only make the sketch look unpolished, but also make it difficult to select requesters’ elements that needs to be removed eventually, because they are fully or partially obscured by the ones that crowd workers created later. P2’s intermediate prototype in Fig. 3.6-a) shows an extreme case where the workers did not delete anything until the requester specifically asked them to. For example, after a worker replicated the outer border of the requester’s mobile app sketch, the worker asked the requester for permission to delete the corresponding part of the requester’s drawing (Fig. 3.5-2).

On the other hand, we noticed that removing pieces of the requester’s initial sketch is problematic as well; it results in the loss of visual cues for recalling past requests and for understanding implicit requirements (e.g., alignment of multiple elements, spacing between elements) that may not be apparent when viewing individual elements of the sketch. In the T1 sessions of the study, the final outcome is a rough, but complete sketch; however, in the T2 sessions, usually workers do not have the opportunity to see a complete sketch like in Fig. 3.2-(T1).

Nearly half of the crowd team actively used a workaround to better handle this problem. In three of the sessions, crowd workers or requesters used a “sandbox” approach to demonstrate or brainstorm UI elements outside the canvas’s whitespace and then place them in the right position when ready, with the removal of the corresponding sketched element immediately before the placement. This was effective in avoiding clutter and confusion with overlapping elements and minimizing the time that requester’s visual cues are missing. Another approach to avoid clutter, as well as to preserve the requester’s hand-drawn sketch, was to use a split canvas. Crowd workers in two sessions used a split canvas, keeping the requester’s drawing on one side of the screen and creating the higher fidelity prototype sketch on the other side of the screen (Fig. 3.6-b). As we discuss more in Section 3.5, creators of future prototyping tools should consider keeping the requester’s initial visual cues separate and protected from the higher fidelity prototyping, as well as making sure the visual cues do not physically interfere with creating the higher fidelity prototype.

3.4.5 Varying Pace of Making Requests

In this kind of collaboration, requesters may communicate their UI requirements to workers in temporally different fashions. Exploring the variability of temporal patterns can help us understand types of challenges that requesters and crowd workers have for a specific pattern, and allow us to consider how future systems could address such challenges. To investigate the temporal pattern of a speaker, we calculated a participant’s pace (speech rate) and saw how it changed over time for all participants. Here, we present two participants’ speech rates, showing two different representative patterns in Fig 3.7-1 and Fig 3.7-2; the slope of the graph indicates speech rate of the person in the particular task (words per minute). When P2 conducted T2 (see orange line in Fig 3.7-1), they started speaking at an approximately constant pace for the first 154 seconds, closely matching the pace when they conducted T1 (the blue line of the same figure). P2 then stayed mostly silent for 10 minutes thereafter (154—764 seconds in Fig 3.7-1.), with only a few spoken clarifications, allowing the workers to catch up on completing requests. In the meantime, P9 spoke at

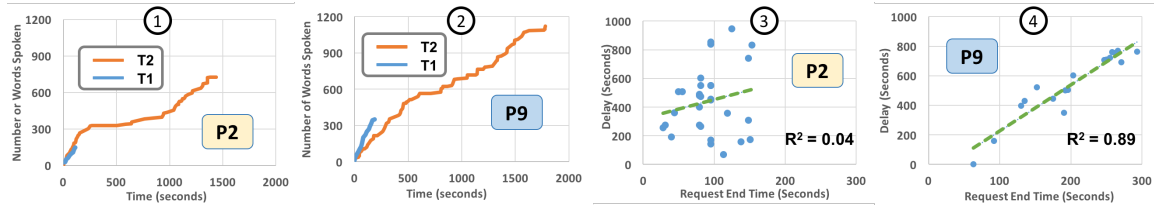


Figure 3.7: (1),(2): the number of words spoken so far(y-axis) over time (x-axis) for P2 (designer) and P9 (non-designer). The orange (longer) line indicates T2 and the blue (shorter) one indicates T1. For T2, P9 speaks in constant pace throughout the session, whereas P2 finished the initial requests quickly and stay silent for a while. (3),(4): response delay over time. (x-axis): the timestamp at which an initial request was made per element, (y-axis): the time (delay) it took for crowd workers to respond to the request. Delay for P9 showed a linear trends, which indicates the requests got quickly backlogged.

an approximately constant words-per-minute rate throughout their T2 session, as can be seen by the approximately linear orange line in Fig 3.7-2. One potential reason that can account for the difference between P2 and P9 could come from their varying levels of expertise. Reviewing screen recordings in detail, we found that P2, a UI designer with 4 years of professional experience, was effective and efficient in describing and creating a rough sketch of the given UI (e.g., better/quicker drawing, extensive use of copy/paste which led to consistency between elements of same type). The sketch gave the workers a complete visual specification that demonstrates the overall structure of the application effectively. In the meantime, P9, a non-designer, manually drew each element of the interface, not only taking longer time than P2 but also the inconsistency between elements may have created confusion to crowd workers.

The ways in which requesters verbally describe and draw can also influence workers' request completion patterns. For example, Fig. 3.7-3 and Fig. 3.7-4 illustrate the delay from time that a participant (P2 or P9) made the initial request for an element to the time that a worker began to work on that particular element in T2. For example, if the coordinate of a data point is (100,20), it means that the crowd workers began working on the corresponding element 20 seconds after it was requested at time 100 seconds. We present data for the first five minutes of each session, as this corresponds to the time that the bulk of initial element creation requests were made and most of the remaining communications thereafter were clarifications about initial requests or corrections to the prototype. The graph for P9 (see Fig. 3.7-4) shows a fairly linear trend between the time that a request was made and the delay for that element, with an R^2 value of 0.89, indicating that many requests got backlogged over time. The graph for P2 (see Fig. 3.7-3), however, shows no such trend; delay for response to a request seems to have no correlation with the time a request was made. Potential

challenges that crowd workers would have had in P9's session include needing to listen to the request as well as work on the prototype and not being able to recall all the requests that were made later in the session as the delay gets longer. This sometimes force a requester to make the same request again. On average, participants made *repetitive* requests 5.0 times during their T2 sessions. On the other hand, crowd workers also asked the requester to repeat instructions from time to time (1.4 times per session on average; see the example message Fig. 3.5-3). Overall, we can see that varying temporal patterns can pose certain challenges for crowd workers and can determine later patterns in the session. It would be worthwhile to consider how such a system can address these challenges for workers, and to design the system to guide requesters to effectively make requests.

3.5 Design Recommendations

Based on the insights we learn from the study, we suggest design recommendations for crowd-powered design tools that involve requesters and crowd workers. We believe these will be applicable to interactive crowdsourcing systems that have a similar structure of user-crowd collaboration, especially in creative domains.

3.5.1 Turn Live Speech into a Structured Task List

Even though the system supports real-time collaboration between a requester and crowd workers, the natures of the two groups' work are dissimilar; in this case, a requester provides a verbal description and workers build the prototype. Due to the asymmetry in communication, requesters will not have a fluent communication close to in-person collaboration and remote conference call. A backlog of requests can build up quickly if the requester interleaves their requests with giving feedback on and correcting existing artifacts. To that end, crowd workers need ways to structure verbal requests and to revisit past requests. It will be desirable for crowd workers to be able to replay the audio for a specific task. In addition, workers should be able to quickly find and navigate what to replay close to a hierarchical to-do list where each item is associated with an audio snippet, as suggested in [175]. This may help requesters reduce the time they spend monitoring the system in real time; workers can replay the requester's verbal instructions, and therefore, hopefully complete more tasks the first time they are asked, with fewer clarifying questions and less repetitive requests. Given the real-time audio streaming, this poses an interesting technical challenge of recording live audio as well as being able to revisit the audio at the

same time, two actions which are usually mutually exclusive. Audio being replayed would mask the live audio.

3.5.2 Address Asymmetry in Communication and Expertise

We found that the asymmetric communication pattern of the requester’s speech versus the workers’ typed text poses a number of challenges. We believe there are certain cases where this asymmetry is inevitable (e.g., live streaming events in social media—Twitch, Facebook Live), but the slowdown of communication in a collaborative setting can be costly in crowd-powered systems. An easy but overly simplistic solution to this problem could be to equalize two different modalities automatically, when possible. For example, spoken communication could be transcribed into the chat box. Furthermore, the developers of such systems should be aware that the potential expertise gap can be a challenge in communication when designing interactive tutorials and recruiting workers (relevant background, prior experience with the task). More advanced NL techniques that can adapt the reference to more general terms can be anticipated in interactive crowdsourcing systems [176]. We believe asymmetry in communication and expertise is an intriguing research topic that requires separate attention in future work.

3.5.3 Protect Requester Artifacts in Shared Space

As previously mentioned, state sharing in crowd-powered systems should occur in different ways, especially given that requesters and workers have different purposes with regard to the shared artifact. In our case, a requester’s sketch being deleted bit by bit over time, as well as the canvas being cluttered, were problematic. Social protocols can be vague, given the temporary and remote nature of the process. This necessitates coordination policies embedded in the system [177]. Particularly, in this design context, we identified two needs for future systems: 1) the requester’s visual cues should remain intact and protected (access control), and 2) the requester’s visual cues should not interfere with the workers’ work area as they create the higher-fidelity prototype (spatial separation) [178].

A simple solution for this is to denote separate prototyping areas side by side on the canvas, one for the requester only and one for the workers only, which is similar to the strategy employed by some crowd workers who split the canvas during their session. Giving different read-write access to requesters and workers will help protect requesters’ visual cues. Alternatively, a requester’s sketch can be visually presented as a transparent overlay on the workers’ canvas, with functionality to easily remove or hide it.

3.5.4 Augment Workspace Awareness of Presence and Implicit Interactions

It seems that the workspace awareness is much more limited, not only because the task involves remote collaboration, but also because of the transient nature of crowdsourcing [156]. Even with the shared canvas and live audio streaming, requesters had trouble in using the system. For example, during the experiment, requesters wanted to ensure their voices were audible to crowd workers. This can be challenging, again due to the lack of immediate (audible) feedback from the crowd workers when they do not respond through the chat box quickly. A simple visualization of the requester's audio as heard on the workers' machines can help reassure the requester that audio is functioning properly on all team members' machines. We also found that non-designers referred to elements by naturally gesturing with their mouse cursors, though not clicking. The deictic gestures made with the mouse cursor did not help crowd workers locate an element, because crowd workers cannot see the mouse cursor's movement unless the mouse is clicked; this information could be useful to crowd workers if visualized in sync with the speech. Gesture traces can be used to supplement the shared visual space [179]. As the canvas is synchronized per-element (as opposed to per-pixel), visualizing the activity level of a worker can enhance awareness of whether they are working on something or are idle. Lastly, crowdsourcing systems are often designed to intentionally mask existing information for privacy, in which case developers need to consider how to preserve awareness in such systems, especially if they should support real-time collaboration [180, 181].

3.5.5 Train Workers to Collaborate, Not Just Use

We believe that the findings from this work should be considered when training crowd workers. Until now, the training for SketchExpress focused on tool usage. However, there are other aspects that facilitate communication and collaboration for crowd workers to understand, for example, acknowledging requests by responding immediately and addressing the requests in timely manner. In addition, there are other important factors that this study did not explore but are essential to accomplish the goal of real-time collaboration, such as task coordination, awareness of tasking, and communication amongst workers. Designing interactive tutorials that teach these concepts will help crowd workers be effective in real-time collaboration.

3.6 Conclusion

The goal of this work was to study the collaborative aspects of an interactive crowdsourcing system and better understand the interaction between requesters and crowd workers in such systems. We conducted a user study that allowed us to understand how real-time collaboration occurs and how requesters understand such a collaboration type. We also explored various aspects of requester-workers collaboration: the style of language they speak and corresponding challenges, the ways in which they work together on a shared canvas, the challenges arising from asymmetric communication channels, and the variability of their communication patterns over time. Our study's findings directly inform the design of future crowd-powered prototyping systems, and more generally, will help reinforce our understanding of the interaction between requesters and online crowd workers, especially in the domain of collaborative creation. The design recommendations we make will help provide a reference for researchers and practitioners who are interested in developing more powerful interactive crowdsourcing systems, as well as mixed-role interactive systems more broadly.

CHAPTER 4

Tools for Live Collaborative Programming³

Software development is a complex task with inherently interdependent sub-components. Therefore, live collaboration between an end-user and programmers poses intriguing engineering challenges in coordinating their efforts in real time. I introduce a form of networked music performance where a performer (a consumer) plays a mobile music instrument (an artifact) while it is being implemented on the fly by programmers (creators). This setup poses an additional challenges in live collaboration given the musical context; the communication channel is limited (e.g., non-verbal or textual communication) and highly dynamic nature of music performance that occurs in extremely short time window (tens of minutes). There emerges a number of issues: (1) the need for effective communication, (2) issues of conflicts in sharing program state space, and (3) remote control of code execution. This chapter proposes solutions to these problems. In the extension of `urMus` - a programming environment for mobile music application development [183] - I introduce a paradigm of shared and individual namespaces safeguarded against conflicts in parallel coding activities.

4.1 Motivation: Live Coding the Mobile Music Instrument

In the era of New Interface for Musical Expression (NIMEs), the development of musical instruments, composition, and performance are often concurrent/out of sequence. For instance, Cook suggest that composing a piece first is a good principle to develop a NIME [184] while Murray Brown et al. argues that concurrent process of composition in instrument building will help convey the music to audiences [185]. Embracing the unclear order of today’s computer music making, I believe deferring the creation of a musical instrument until the time of performance can push the level of the liveness of a musical performance.

Another motivation of the on-the-fly musical instrument building is the fluidity of the concept. Magnusson defines “composing an instrument” as a process of designing constraints

³Portions of this chapter appear in [57, 59, 182]

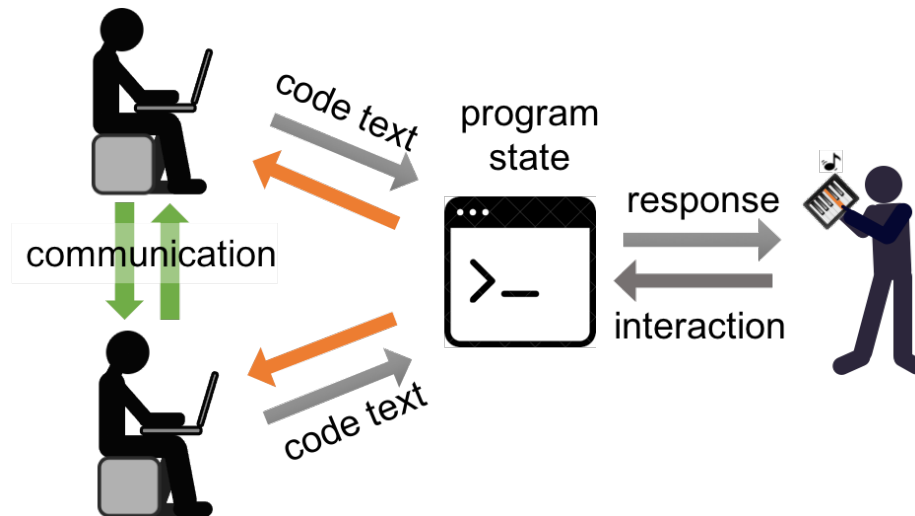


Figure 4.1: In this performance practice, two programmers develop an application in real time by sending code text to an application that is running on a tablet over a wireless network. The application, which is a mobile music instrument, is used by a musician at the same time. While programmers can write and send code to modify the program state, the program state that changes by their own code, collaborators' code and user's interaction is not clear. This obscurity is an obstacle to live collaboration. I suggest a tool that makes the program state visible to the programmers, depicted as orange arrows in the diagram.

for a musical space [186]. Therefore, the act of live collaboration between instrument builders and musician would facilitates impromptu creativity given the dynamic changes of constraints. The motive of a dynamic affordance/constraint of expressivity can vary. For example, it can be a compositional decision of an instrument builder in collaborative improvisation, while, in a different scenario, it can be adaptation (or confrontation) in response to a particular performer's play style (e.g. Jazz instrument player vs. Live looping player). In another case, as already explored in [187, 188], the instrument can be utilized as a device for audience participation where the instrument provides progressive expansion of expressive space based on the learning curve of audience members.

In this performance model, programmers takes role of a composer, instrument builder and collaborators, whereas an instrumental player takes the role of a user who performs the dynamically changing instrument. I believe this distributed model will benefit the aesthetic framework of instrumental music from the fluidity of live coding. In addition, as live coding has focused heavily on audiovisuals, I wish to make an expansion of the field to user interaction setting. As already anticipated in [189], this will bring a set of research questions of on-the-fly instruments, such as playability for performer, which I will explore later in this chapter. For more detailed description on the musical motivation behind this work, see [57].

4.2 a Programming Environment for Collaborative, Live Coding

The question of how to make a collaborative, live coding environment facilitate this mode of networked collaboration is what gave rise to the goal of this work. To that end, I extended an existing programming environment for mobile music - urMus [183]. The code editor of UrMus is implemented as a web application running on a mobile phone so that a user can code on any web browser (usually running on a laptop) and transmit code over a local wireless network to be interpreted on the device. In this remote programming environment, multiple coders can implement a mobile music instrument together on the fly. An example of two live coders and one instrument performer is depicted in Figure 4.1. Through a course of rehearsals and the performance from the previous work [57], a number of improvements was suggested to the environment so as to support collaboration and communication in this setting. The design goal of the extension is to improve uMus so that it will support live collaboration between a musician and programmer in the performance setting. The remainder of this section describes a concrete extension to the programming environment.

4.2.1 Centralized State Space

Wang introduced two models of collaborative audio programming space-server-client and peer-to-peer [190]. These terms are also related to the classification of network music in *centralized* and *decentralized* approaches that Weinberg suggested [191]. In a centralized approach, there is only one program state space that generates the outcome of live coding. Live coding musicians connect to the machine and remotely execute code over the network. Since only one state space is running on one machine, no additional process is needed to either share information or synchronize clocks. Whereas in a decentralized approach, there exist multiple machines used by multiple live coders. In this case, each machine has its own state and generates an individual outcome. Therefore it requires a separate mechanism to share information between live coders (e.g., additional server application or distributed memory). urMus inherently supports a centralized model. Live coders on laptops (multiple clients) can connect to the mobile device (server) and transmit code text to the device wirelessly. Figure 4.2 demonstrates the system architecture of this work. UrMus is based on lua [192], a light-weight interpreter language so that whenever code is run, the code will be evaluated and executed on top of the current state space thus far accumulated (see Figure 4.2-a.) For example, if a live coder submitted code that assigned a value to a string variable named *str*, any live coder who submits code afterwards will have access to *str*. Note

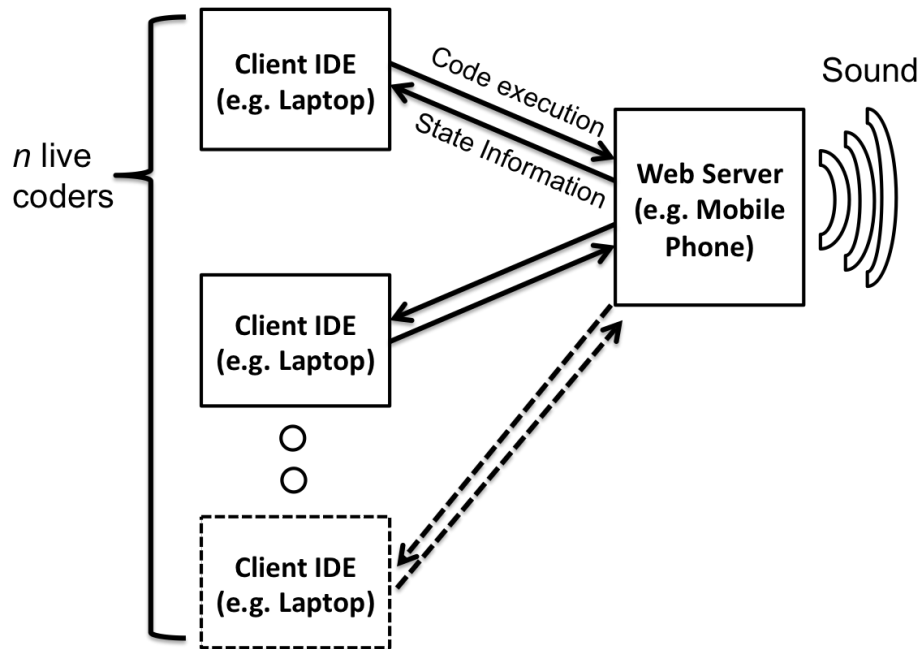


Figure 4.2: Centralized architecture of collaborative, live coding in urMus

that although the editors is web-application, this is different from other environments, i.e., [99]. Here the web-editor on the laptop is a dumb terminal that holds code text and the mobile device is the machine that synthesizes sound and renders a graphical user interface.

4.2.2 Individual and Shared Namespace

In the last example from the previous section, I can find a problem when multiple programmers code in a centralized state space. What if someone else creates a variable named *str* without realizing a variable with such a name already exists? See another example in Figure 4.3-(a). For the collaborative live coding environment with a centralized approach, it is desirable to control this risk of naming conflicts, otherwise a collaborator may accidentally overwrite the state that someone created. One non-technological solution to this problem is live coders continually paying attention to other peoples' code text and being careful not to make conflicts as a part of live coding practice. This introduces extra cognitive loads for live coders and limits the improvisational aspects of live collaboration. Another naive solution to this is to make each live coder have individual state space which will prevent conflicts (see Figure 4.3-(b)). However, this solution obstructs basic ways to collaborate by isolating a live coder from the centralized state space. For instance, a live coder in an independent state space cannot perform critical network music gestures such as *Borrowing and Stealing* [106].

Our solution to this issue is to give each individual live coder his/her own namespace

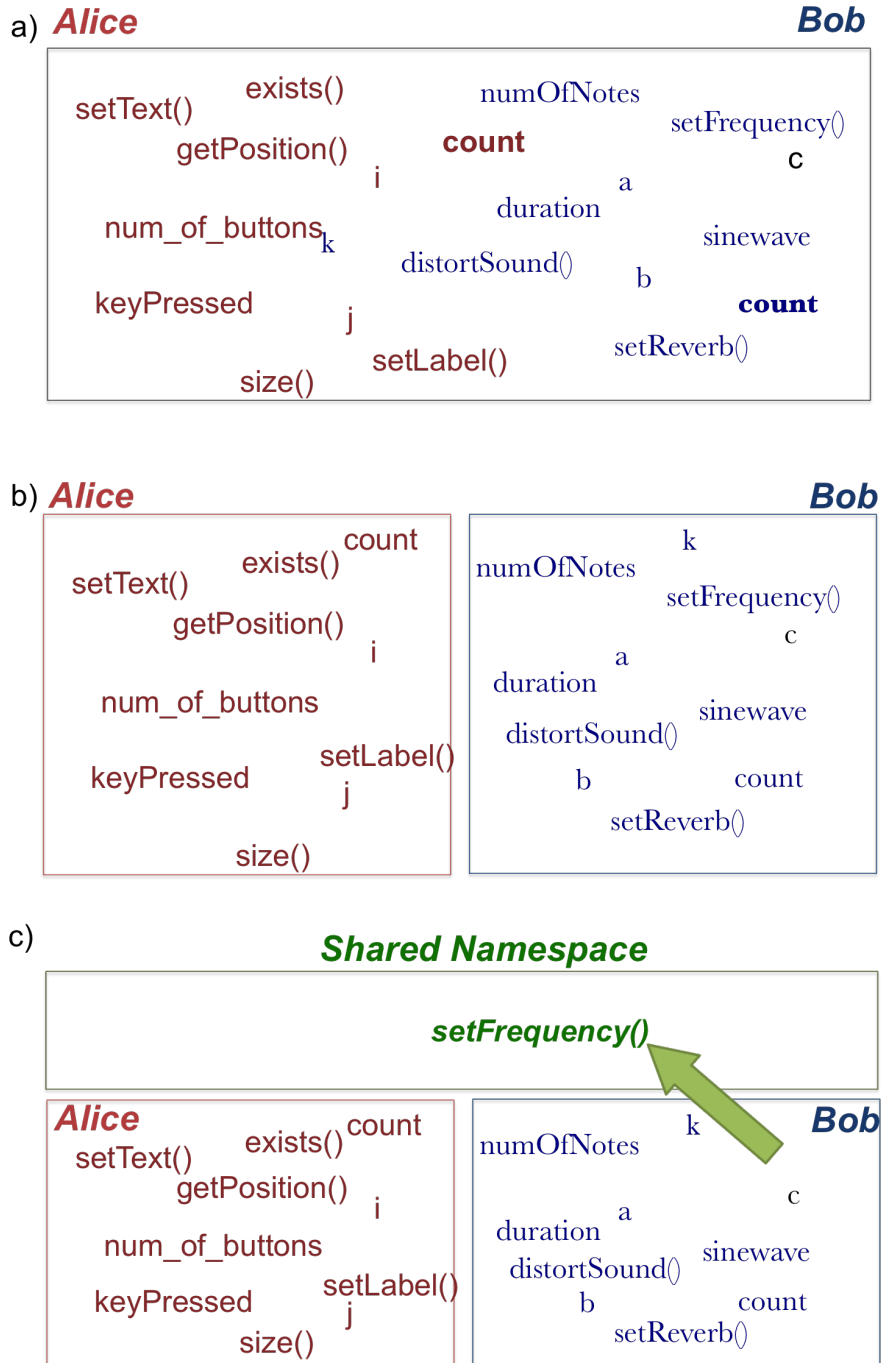


Figure 4.3: a) There is only one state space leaving open the risk you inadvertently modify someone else’s state space. See there are two count variables, the later produced of which will overwrite the earlier one. b) Having a separate namespace per each live coder will prevent this collision but then how will they collaborate without shared state? c) There is a shared namespace that all live coders have access to. A live coder can share either a variable or function to be shared with other live coders. For example, Bob can choose to share a function “setFrequency” so that Alice can execute that function.

and to create a shared namespace separately that everyone has access to. Each live coder can selectively transfer any program state (e.g., variables, functions) of his/her own to the shared space (see Figure 4.3- (c)). In this way, live coders need not worry about someone else corrupting their code by mistake. Note that this approach comes at a price of additional user inputs to select a set of state to be shared. For example, borrowing is possible but stealing is not without user's permission; one has to ask the owner to share a certain state. This structure may facilitate communication between programmers. This is contrary to an alternative where information is open to anyone but the environment only alerts a live coder whenever there is a collision. While urMus requires a coder to make explicit decision on which states to share rather than react to system alerts, the other option also has its own strength in its open structure. For implementation, I utilize lua's functionality of specifying a namespace (or environment in lua term) with a function called *setfenv()*. Whenever a live coder connects to the mobile device, the mobile device assigns a unique namespace for each live coder. Therefore, whatever code executed by a live coder will change the state within the namespace that is associated with the live coder. urMus searches the shared namespace in case that the code could not be evaluated within an individual namespace so that each live coder has access to his/her own namespace and the shared namespace.

4.2.3 Live Variable View

Live variable view provides a live view of the program state that changes dynamically by a programmer's code execution, other programmers' code and the user interaction. Live variable view is similar to other programming environments (e.g., Matlab) that show you a list of variables and their values. It is also similar to the debugging mode in a programming environment where you can see the state of the program at a certain point. However, live variable view is different in a way that it shows separate views of one program states per each connected programmer and the program needs not be stopped as in debugger. The live variable view displays variables, functions available in the state space at the moment, and expressions a programmer specifies (See Figure 4.4),

In Figure 1.5, the live variable view is located in the top left corner of the editor. If a variable is abstract data typed (e.g., array, urMus GUI widgets), it shows all the elements inside the container in a hierarchical tree structure. An expression can be any code text that can be interpreted and returns a value such that you are able to evaluate any information you are interested in during the live coding session. The state space in live coding can change over time for many reasons, including code execution on-the-fly, time-varying variables such as audio data, and user interaction influencing the program (such as pressing a button).

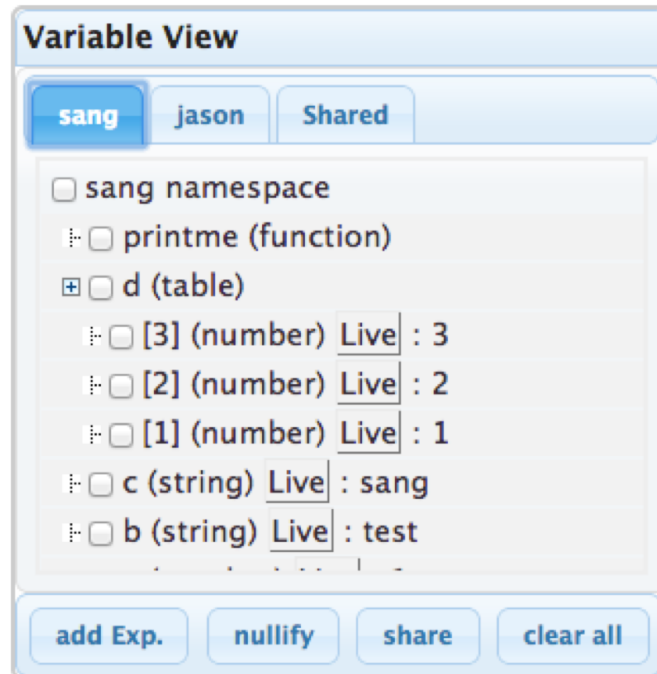


Figure 4.4: Live Variable View. Individual programmers' live program state are displayed, and they can share selected objects with collaborators.

The view updates, by default, any value changed by code execution. The live button right next to each entry can be used if a programmer wants to see the value of the associated entry in real time. For example, if there is a variable that contains audio sample data, one can see the actual sample value in real time. As stated in the previous subsection, there are as many individual namespaces as there are live coders plus the shared namespace. Live variable view will have each namespace presented in each tab. Figure 4.4 shows three tabs are available in the view. Also handled in the live variable view are sharing variables, functions or expressions. One must simply check the checkbox in front of an entry that one wants to share and press the share button at the bottom. Once shared, the entry will appear in the shared namespace tab and any live coder will have access to any entry in the shared namespace.

Having live update of variables (or expressions) in live coding helps communication in two aspects. First, it provides a live view of the program state constructed and shared by each individual. The benefit of this is that it provides a summarized viewport of the program's running state and supplements code text sharing, which does not easily reflect the current state. Furthermore, live variable view enables additional communication with the instrument performer on the mobile phone. In practice, it is important for live coders to

have visual feedback from the instrument performer so that live coders understand the play of the performer, which might affect a live coder's decision on how to shape the musical instrument. Live variable view can offer a flexible monitoring tools of user interaction. For example, one can create a simple function that returns a type of musical gesture that the instrument performer is involved in and add the function as an expression (e.g., clicked, dragged, waved, etc.)

Implementation of live variable view is a composite result of web technologies such javascript/jQuery/AJAX and modification on urMus lua API. It keeps track of all variables and functions declared using lua meta-method and meta-table. At the time of code execution, the live variable view makes a request to retrieve a list of variables that are newly created so that live variable view keeps the list of all the variables in a namespace. Immediately after updating the list of variables/functions, the web editor iterates all the entries in the view and retrieves the value of each variable (or expression) from the corresponding namespace. A functionality is added to urMus application to transmit any data in XML format to the web editor of client. Optionally, all the entries with live button pressed will be updated continuously; otherwise values are only updated per code execution. The entries with the live button pressed will be updated with continuous polling from the state space. I purposely design live update of variables/expression as an option (like a check-box) so that it will minimize the net- work traffic and unnecessary performance degradation in a mobile device by updating all variables/expressions listed.

4.2.4 Distributed Code Execution through Chat Communication

Utilizing textual chat is a common way to support communication not only in live coding ensemble [94, 57, 99, 190] but in many other groupware. The chat window in urMus integrates numerous aspects of collaborative, live programming by making the chat window informative similar to what information one obtains from a console in a programming environment. Through a live chat window, live coders will be notified when someone executes code, shares a variable, or rejects a code execution. All the chat and log messages appear on the mobile device as well, so that the instrument performer is included in the communication loop and receives the notification.

In addition, live chat enables code execution triggered by the networked instrument performer. In other words, any code typed into the chat window will not be executed immediately but will create a message window on a mobile device so that the networked performer can instead execute the code. As the system delegates the networked performer to execute the code, the performer can decide when to pull the change; otherwise, a code

change at an arbitrary moment could interfere with current interaction on the instrument. The message window will state the name of the submitted function and display two buttons - run/reject. The instrument performer simply presses one of the two buttons and runs (or rejects) the newly submitted code. With this distributed code execution, the instrument performer can participate in the coding process by deciding when to execute and having the option to reject. Note that the decision will be fed back to the live coders and will influence forthcoming changes. For a live coder to distribute code execution to another live coder is easy. One can write code into a subroutine function and simply share the function with others. The other live coder can then run the function with one extra line of function call. This will be useful when one wants to synchronize execution of more than one person's code.

4.3 Exploring Real-time Coordination Tools for Ad Hoc Programming Teams

The idea of having collaborative state of programmers more visible can be extended to general programming practice. To extend the idea of live visibility in programming environment, a user study was conducted for transient software teams in existing collaborative programming tools: a version control system and a real-time shared editor. Note that this is not necessarily live collaboration that involves consumers, requesters or end-users. Based on our findings, a shared programming environments is suggested to help teams effectively self-coordinate on their task.

4.3.1 Ad Hoc Crowd Programming Teams

Software development is a complicated process that frequently requires a diverse range of skills and insights. Crowdsourcing has the potential to make software production more flexible, scalable, and efficient, but this is difficult due to the inherent interdependency between sub-components in software code. As a result, coordination costs grow significantly as team size increases [193]. In this section, I motivate an approach to coordinating ad hoc teams in which workers can collaborate to write code remotely in real-time so that the team can complete more complex and moderately defined tasks quickly.

Crowdsourcing recruits groups of workers through an open call [194], and has been used to complete complex tasks [195, 155], and even continuous real-time tasks [196, 117]. Prior work in crowdsourced software engineering has leveraged the crowd using a number of different models [197]. For instance, Topcoder leverages a community of programmers using

a competitive model where contributors participate in programming contests [198]. Latoza et al. suggest a systematic approach to decomposing a complex programming task into a set of microtasks that can be quickly solved by individual crowd-workers [199]. However, such approaches add overhead, both at the initial stage when preparing well-defined tasks, and later when results need to be integrated [200]. Ad hoc teams leave task decomposition and delegation to workers themselves, but without structured coordination or workflow design, can be inefficient [201].

Additionally, online programming assistance services that simulate remote pair-programming (synchronous) [202, 203] and services that provide programming assistance integrated into version control systems or workplace collaborative platforms (asynchronous) [204] have launched in the last few years. Codeon realizes asynchronous crowd-assisted programming through an on-demand support model, improving developer productivity by 70% over state-of-the-art tools [63, 205]. As these platforms mature, it is important to explore rich and efficient ways to interact with crowds of programmers. Our work anticipates new assistance platforms that take advantage of the scalability of crowds while maintaining the benefits of tailored support.

4.3.2 Coordination Costs in Ad Hoc Teams

Modern programming environments support collaborative programming in various ways. Version control systems, such as `git`, are widely used in collaborative programming as programmers can work in distributed manner and synchronize easily with the main code repository. In this case, resolving merge conflicts requires additional effort and communication, often making programmers move away from collaborative programming projects, especially given the short time-span in the context. Collabode [206] introduced a system that addressed the issue of breaking the collaborative build without introducing the latency and overhead of explicit version control. I conducted an experiment to further understand the coordination issues and costs that arise when groups of programmers are asked to complete a programming task without clear individual sub-goals.

At the software level, real-time shared environments [207, 208] can help mitigate much of the coordination costs between workers because: 1) the system only maintains one master copy, so individuals need not worry about code integration, and 2) the most up-to-date code is visible to everyone, meaning that workers can prevent potential conflicts and redundancies more easily before they propagate. Web-based IDE tools, such as Koding [209] and Cloud9 [210], enable users to code collaboratively in real-time. However, the needs of coordinating task decomposition and delegation are left up to users. Furthermore, allowing

simultaneous access to a shared resource can cause new problems, such as corrupting someone else code. While these challenges can be addressed with in-person communication and organizational efforts (roles, responsibilities, team structures) in traditional software development teams, the inherently temporary nature of crowd teams necessitates additional tools and methods for self-coordination. To identify challenges in coordinating ad hoc teams, a user study is conducted with two widely used types of collaborative programming tools: version control systems (VCSs) and a shared editors, each accompanied by a call/chat (Skype).

4.3.3 Identifying the needs of self-coordination

4.3.3.1 User Study: Ad Hoc Programming Teams

This study simulates a scenario in which an end user developer hires crowd workers to form a small ad hoc team (2-3 people) and to complete a short programming task in an hour. The study had three conditions: **C1**, individual programming (1 programmer); **C2**, programming in group on a shared-code editor (2-3 programmers); and **C3**, programming in group with support of a version control system (VCS). All participants used an IDE (atom.io), and the participants in (**C2**) used a plug-in (atom-pair, <https://atom.io/packages/atom-pair>) that synchronizes code text, while the participants in (**C3**) used a version control system (git) in addition to the editor. Two collaborative groups (**C2**, **C3**) were also connected through Skype.

I recruited 12 participants (from authors' university (7) and UpWork (5)) and conducted two sessions per condition (E1-E6; refer to Table 4.1). Every participant had more than a year of web programming experience, and are either a freelancer or a senior undergraduate student. Each participant was asked if they were familiar with the programming concepts necessary to solve the task (regex, event handlers, and selectors). Participants were asked to complete a task in a group of two or three or independently (max time: 60 minutes). The task was to create a simple web application that takes a text content and evaluate the readability of the content by calculating various statistics (word count, letter count, five extra readability index). The task can be decomposed into a set of subtasks easily and they are dependent on one another or share common functionalities.

Participants in a group did not know each other and were asked to work on the task collaboratively with no guidance as to how to collaborate beyond using the designated tools. All participants were connected to the experimental session through the conference call and were asked to record their screen. In the end, each participant was asked to fill out a survey that has a set of open-ended questions about the collaborative programming

Condition	individual(C1)		shared editor(C2)		version control system(C3)	
Experiment	E1	E2	E3	E4	E5	E6
Number of Participants	1 (W)	1 (W)	2(S,S)	3(S,S,W)	2(W,W)	3(S,S,S)
Time Taken (in min)	60	60	60	58	60	60
Evaluation (max 100)	80	55	69	75	73	71.5

Table 4.1: Six experiments (E1-E6) are ran in different conditions. For condition 2(**C2**), participants used a shared editor and for condition 3(**C3**), participants used a version control system (git). **W** indicates a crowd worker and **S** indicates a university student.

experience. Code results submitted by the teams were evaluated based on how many test cases the program satisfied, as well as the authors’ assessment of the code quality. After the experimental sessions, I analyzed the screen and voice recording to identify all of the communications between programmers. I also analyzed the effort spent coordinating the team’s efforts in two different environments during the session.

4.3.3.2 Result: Shared-code Editor vs. Version Control System

It is worth noting that the goal of the study is not to confirm if one of the conditions outperforms any other. Rather, the goal is to identify incidents where programmers coordinated their efforts and to collect participant feedback from the survey.

Two collaborative conditions necessitate different coordination efforts. For the version control system condition (**C3**) in which code text was not shared in real time, two groups took opposite approaches to their collaboration. The first team (**E5**), composed of two crowd workers, split the work initially, wrote code in parallel, and merged individual code at the end. There was minimal interaction between the two programmers: it was limited to task distribution in the beginning and for code integration at the end. The consequence of two programmers working in parallel was JavaScript code in two different styles, i.e., one used regular expressions with jQuery, while the other used character-by-character comparison using arrays in pure JavaScript. While this did not hurt the correctness of the code, the style of the code in a file was not consistent which may lead to higher maintenance cost in the future.

On the other hand, the second team (**E6**) chose to communicate actively from the beginning and discussed how they could avoid merge conflicts when pushing code to the repository. They chose to create a JavaScript file per subtask, which complicated the coordination process and added the significant overhead of time (40% of total time). The first team (**E5**) spent 21% of the allotted time splitting the work into two subtasks, updating/merging their code with others, and testing the merged one. During merging the code, only one of

two programmers was working in testing and validating the code. Similarly, team (E6) also had moments where a programmer asked others to wait and not to commit any code until they pushed the code. While two VCS groups chose different strategies for collaboration, I found they ran into the common bottleneck: task completion was deferred by configuring collaboration in the beginning and merging code at the end. The time it takes to coordinate collaboration in version control systems would have been significantly less if they were using the shared code editor.

For real-time code sharing condition(C2), I observed that maintaining single global “live” copy of code facilitated collaboration; this allowed participants to have access to more information, which results in more consistent code and initiates communication. They expressed the benefits of reading someone else’s code in real time; (E4-2) wrote that they “*avoided looking into online docs for some details*” and (E4-1) noted that “*the other programmers thought of a code organization that I didn’t think of.*”). On the other hand, some people expressed that they felt “distracted” (E3-1,2) as they cannot test their code due to the incomplete code of others. This problem of being corrupted by code-in-progress in a shared editor has been addressed in [206]. However, the style of the code from (C2) was evaluated to be stylistically more consistent and readable than the ones from the (C3) group, leading to less cost for later integration and maintenance [211]. In general, I see that the advantages of using a real-time shared editor outweighed the technical difficulties in its performance and testing. Also, both groups in (C2) spent time in coordinating task decomposition, which potentially explains why the durations taken in the collaborative sessions are similar to the ones in the solo session (C1).

4.3.3.3 Result: Needs for Communication and Awareness

I discovered that the level of communication could be drastically different per group. The lack of communication can be attributed to technical issues as well as social norms (language barriers and lack of familiarity with strangers’ coding styles). For example, it took 12 minutes in a session (E5) to split the task into two parts and the participants never communicated to each other except when merging code into the repository towards the end, resulting in the inconsistent style of the code. I found from the videos that the groups who did not communicate actively faced further issues (e.g., wasted time on redundant tasks or inconsistent code). One participant (E4-3) commented that “*it would be much more efficient if I knew each other due to better communication,*” and (E5-1) responded that the task “*should have been reviewed and discussed in depth beforehand to determine the dependency of tasks.*” While the level of communication can vary depending on the different factors, the potential lack of communication necessitates nuggets of information that will help initiate

and facilitate communication among programmers.

Further, participants expressed the needs of awareness in the task distribution and its progress. Participant (E4-2) commented that *“what was difficult is to understand who does what at this moment.”* and (E5-2) wrote that they would like *“a system that would monitor tasks that the programmer is busy with and distribute this information to the other users.”*. Various features to support simple awareness are used in shared environments [84, 151], and typically highlight edits (or cursors), which files are active, and users’ connection status [208]. While such awareness features are useful for determining the spatial location of cursors, or which file is being edited, they are not sufficient to provide high-level information on the task distribution and overall progress. Simple awareness features could also mislead collaborators – e.g., the location of an inactive cursor while a programmer searched for online materials made one participant confused that it was *“difficult to determine if anybody is editing some functions in real time and decide if I can edit it”* (E4-2).

Finally, I found that early assignment of multiple subtasks to individuals can lead to a potential bottleneck that makes part of the group wait on a programmer to complete their subtasks. In two collaborative sessions (E4, E6), I observed that participants realized (as they wrote code) the dependencies among their subtasks, and then determined that they needed to change their assignments on-the-fly or wait for others to finish certain subtasks. The potential workaround to this problem is to assign only one task at a time so that the interdependency of sub-components that emerge later can be easily handled.

4.3.4 A Shared Code Editor for Ad-Hoc Teams

Based on the initial insights from the analysis of our experiments and survey results, I find that the following design elements would help reduce the coordination costs in a collaborative programming environment for ad hoc teams:

- a shared-code editor that avoids multiple versions
- displaying information on coordination that facilitates communication among team members
- self-coordination tools for programmers to flexibly complete tasks and provide progress awareness

To address these issues in communication and coordination, I am currently developing a shared code editor that facilitates self-coordination and communication (Figure 4.5). I introduce a subtask view that will help programmers self-organize their work below in

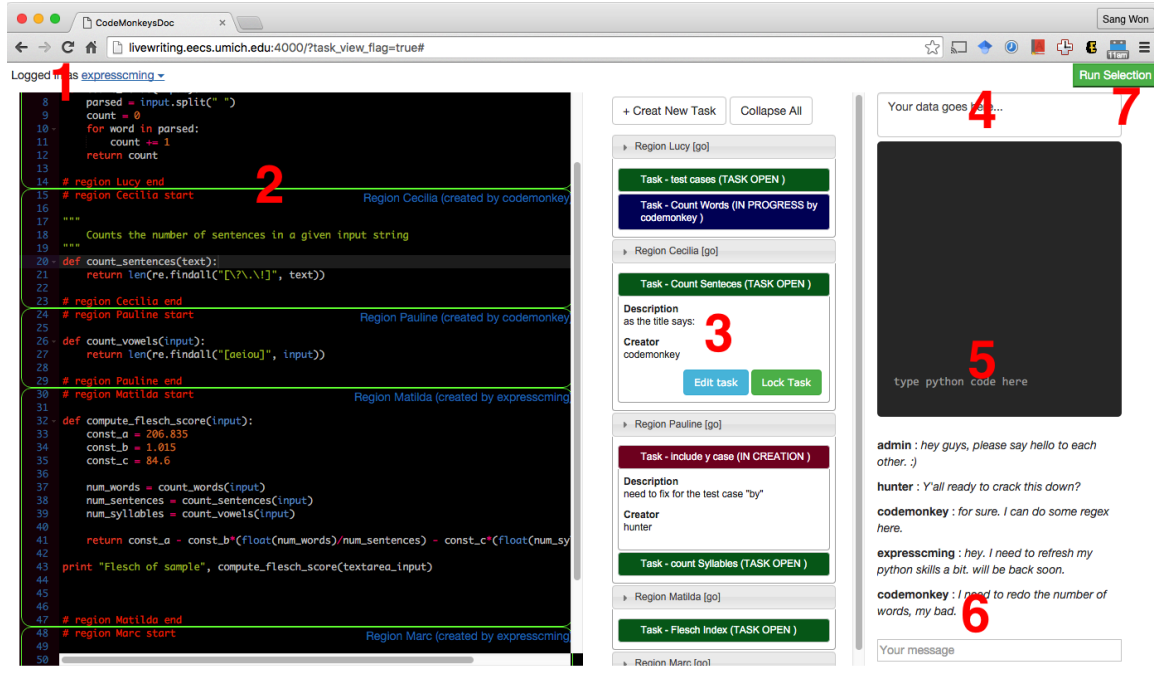


Figure 4.5: a shared editor with subtask view. 1) log in 2) a shared editor with region boundaries 3) subtask-view 4) input data form 5) console output 6) chat interface 7) run button

addition to basic functionality like code synchronization, log in/out functionality, a chat interface for real-time communication, and an integrated run-time environment.

4.3.4.1 Subtasks as a Communication Channel

The subtask-view (Figure 4.5-3) enumerates the list of subtasks that are created by programmers. Programmers can define a subtask and associate a region in the code editor with the subtask before writing any code. Therefore, one should declare what the subtask will be about first, before writing code. Information about subtasks, such as the title and description, are shared in real-time as they are entered. This real-time information serves as a means of declaring the sub-components of the code and indicating how the overall task is decomposed into a set of subtasks up to the moment. While I apply this method to one file and the unit of subtask is a region, note that the unit can be bigger depending on the scale of a project, programming languages being used, and its programming environment. This can help programmers understand what subtasks are defined and coordinate their roles in the session by reviewing existing tasks and creating new ones. In addition, forcing programmers to enter a title and description also helps document the code. Creating subtasks is the primary way to decompose tasks, and anyone can take that initiative.

4.3.4.2 Task Distribution by Locking Subtasks

In addition to the subtask view being a declarative and communicative medium for crowd workers, locking mechanisms of subtasks help programmers be aware of the task distribution status. To be able to write code in a region of the code editor, a programmer needs to lock a subtask that is associated with the region. The locking mechanism is designed to be exclusive so that one can only lock one subtask (and thus one region) at a time. Three states of a subtask (locked - blue, available - green, or in-creation - red) represent what tasks are available to prevent potential conflict. Since creating a subtask is separate from locking the task, task distribution will be delayed until the moment a programmer locks the task. Assigning subtasks is done by individuals, making the process of writing code a part of the system's self-coordination mechanism.

I have presented initial results from a user study of ad hoc team programming to understand coordination costs in collaborative programming environments for crowd workers, and proposed a collaborative programming environment that facilitates self-coordination and communication. A set of interesting challenges remain as a future work: e.g., recruiting expert crowd workers, run-time environments for the shared editor, and code refactoring across regions.

4.4 Conclusion

In this chapter, I have explored two important topics in live collaborative creation — 1) ensuring visibility of a shared artifact that are often hidden in a groupware for remote real-time collaboration, and 2) designing interactive systems to coordinate collaboration to avoid conflicts and facilitate task decomposition.

CHAPTER 5

SketchExpress: Crowdsourcing Interactive Behaviors in GUI Sketch Prototype⁴

5.1 Introduction

Low-fidelity prototyping at the early stages of user interface (UI) design can help designers and system builders quickly explore their ideas. However, interactive behaviors in such prototypes are often replaced by textual descriptions because it usually takes even professionals hours or days to create animated interactive elements due to the complexity of creating them.

In this chapter, I explore ways in which non-expert creators can participate in live collaboration with UI designers in order to prototype interactive behaviors in early GUI sketch. SketchExpress, a crowd-powered prototyping tool, enables crowd workers to create reusable interactive behaviors easily and accurately.

The SketchExpress is built upon Apparition [155], a system that leverages the online crowd to create interactive behaviors for the designer. The complexity of behaviors that crowd workers could demonstrate in Apparition was limited. In addition, the manually demonstrated behaviors are ephemeral and do not persist with the sketch once the live collaboration session is over.

In SketchExpress, designers—*requesters* in this crowdsourcing context—verbally describe their prototype and one or more crowd workers collectively produce a corresponding sketch. A requester can draw content and then describe aloud desired behaviors without having to stop to implement/create the functionality they are describing. Behind the scenes, non-expert crowd workers listen to the verbal requests and use SketchExpress’s UI to create replayable animations in a matter of minutes. Designers can also mock-up interactive behaviors using SketchExpress without the crowd, but crowdsourcing allow the system to make the creation process fluid and quick, which in turn makes requesters’ interaction with

⁴Portions of this chapter appear in [60]

the system minimal and natural.

SketchExpress does this by introducing the *demonstrate-remix-replay* method, which is easy to learn and expressive enough to prototype complex behaviors. Based on the requester’s verbal description, crowd workers first *demonstrate* a behavior that is recorded by the system as a series of operations. Workers can then *remix* the recorded animation to further refine it. Once this is done, any worker or designer can *replay* and compose multiple animations with a click of a button, making it possible to effectively support complex (multi-part) animations in early-stage prototypes. The resulting prototype retains complex behaviors and can be used to iteratively explore design ideas, communicate with collaborators, and act as a “living spec” for future implementation.

The contribution of this work is as follows:

- a novel method, *demonstrate-remix-replay*, with which non-expert crowds can prototype interactive GUI behaviors;
- SketchExpress, a system that creates reusable, higher-fidelity animations in early sketch;
- validation of our approach through a user study with crowd workers recruited from Mechanical Turk.

The results contribute to the broader goal of creating a prototyping tool that helps anyone design and/or modify GUIs. SketchExpress aims to provide computational tools that non-experts can participate in collaborative creation in live settings. In addition, the system will help non-expert designers (consumers) to rapidly iterate on ideas and hand-off tasks to crowd workers: non-experts to participate in the design and improvement of software systems; researchers to quickly mock up interactive tools for experimentation; and students to create engaging examples even before they learn to program.

5.2 Challenges in Prototyping Interactive Behaviors

Creating interactive behaviors in early stage prototypes is challenging for multiple reasons. First, interactive behaviors involve the dynamic transformation of multiple interface elements, which indicates that they cannot be easily presented in static images. For example, to demonstrate how a user can “swipe to unlock” a smartphone screen, a static sketch is not enough. It requires a description of cause and effect behaviors, e.g. what happens to the button with the arrow moving within the rail when it reaches the right end, what happens when a user releases the button halfway.

Existing tools typically provide a set of predetermined behaviors that one can choose from. These preset behaviors tend to only support specific types of applications (e.g., transitions between web pages, or drop-down widgets) well. As a result, these tools are limited to prototype behaviors in general systems that go beyond traditional window-based GUIs. For example, animating the behaviors of a video game character in a side-scrolling game (e.g., Super Mario) or how the enemy characters (e.g., turtles in Super Mario) respond to the other element's changes (e.g., Mario bouncing on them) cannot be accomplished easily with existing tools. Existing tools that can support expressive interactive behaviors require expertise – and even then, creating high-fidelity prototypes can take hours or days even for professional designers, which makes them inappropriate for use in the earliest stages of UI design. Interactive prototyping requires first learning these professional tools, making it difficult for non-experts (i.e. UI end users) to participate in the UI design process.

SketchExpress builds upon prior work on: 1) prototyping tools that make dynamic and interactive sketches and 2) end-user programming by demonstration systems. We discuss prior work in these domains to provide context for the design choices made by SketchExpress to help make early stage prototyping of interactive behaviors more accessible to both non-expert designers (requesters) and workers. We also review existing tools that permits users to create interactive behaviors in GUI prototypes.

5.2.1 Designing Interactive Behaviors in Sketching Tools

UI prototypes are used by system builders to explore new ideas in depth more quickly. Rather than building fully functional systems from the beginning, prototypes permit quick trial and error iterations that can be easily produced and evaluated. Systems like SILK [212] and DENIM [213] were early efforts that reduced the overhead of prototyping by recognizing designers' sketches as interface elements and implementing the idea of wireframing, respectively. However, the outcome of such tools is most often a static sketch that does not include the interactive aspects of the GUI. Designing UI behaviors is harder than designing layouts because the behaviors are more complex to demonstrate and the tools available to designers have more limitations [214]. There are several professional UI prototyping tools that can be used to program interactive behaviors, and though these tools have become more user-friendly, they are still difficult and time consuming for non-experts to learn and use. Additionally, these tools often support only a limited set of animations in specific UI contexts (e.g. wireframe transitions, standard widgets for mobile applications), which makes it difficult to prototype interactive behaviors for general applications. We will discuss existing tools more later in this section. SketchExpress makes the creation process natural and expressive,

recruiting crowd workers "power" interactive sketches using the demonstrate-and-remix approach without spending extensive time learning how to use complex tools.

Previous research has also created tools that support dynamic sketches and are easy to learn. For example, non-expert users were able to learn K-sketch within 30 minutes and use it to generate dynamic illustrations that can be played as an animation within 7 minutes [215]. SketchExpress draws on a number of important ideas about recording demonstrations, dubbing, and post-edits from K-sketch and other similar systems [216, 217]. One important distinction between SketchExpress and K-sketch is that K-sketch creates a single, linear series of actions to represent a behavior (as if it were a video), whereas SketchExpress generates a set of animations that can be replayed independently and simultaneously, allowing workers to mix and match existing actions to represent new behaviors. Thus, SketchExpress prototypes can end up in various states depending on which combination of animations are executed.

Alternatively, Sketchify lets designers generate completed interactive behaviors through a scripting language [218]. Users can write scripts to configure subtle relationships between elements and interactive materials (e.g., sensors), focusing mainly on the interactivity and integration with other input sources. Their study showed that scripting "does not fit" the overall sketching system and it also confirms our belief that too many functions "may cause confusion and overload" [218]. SketchExpress transforms a static sketch into an animated prototype without programming and leverages human computation to handle aspects that would otherwise require script logic.

More recently, Kitty employed various methods to enable a dynamic relationship between elements on canvas [219]. While Kitty was developed for artists to create illustrative animations, SketchExpress utilizes a more traditional sketching tool. The process of creating animations in Kitty is close to programming, using: kinetic textures, relational graphs, and functional mappings. However, to crowdsource prototyping we need much simpler yet similarly expressive interaction techniques that non-expert workers can pick up nearly instantaneously.

Most existing tools attempt to simplify the programming process of interactive behaviors, retaining the logic behind the behavior. In contrast, SketchExpress records manual demonstration and lets crowd workers remix it to refine the behaviors. This Wizard-of-Oz approach has been shown effective in making the design process accessible to a broader population [220, 221], but has been used on static UI sketches rather than interactive components. An adaptation of the Wizard-of-Oz approach where human operators manipulate paper prototypes to show interactive components has been traditionally used to demonstrate the dynamic behaviors of prototypes [222]. Animating physical mockups is a widely used

and powerful technique, but it is limited by the physical efforts needed to produce the cutouts and to manually animate the objects. For example, the materiality of physical mock-ups makes some types of behaviors difficult to demonstrate (e.g. scaling, re-coloring, opacity). In addition, using videotaping and editing to replay demonstrations of physical mock-ups can be used, but again yields a single linear progression of actions, in contrast to SketchExpress’s recomposable animations that result in greater expressivity. Lastly, SketchExpress has the advantages of electronic sketching (discussed in detail in [212]). For example, an electronic sketch can be quickly drawn, easily modified using operations (i.e., *save*, *copy*, *paste*, and *edit*), and shared with others (e.g., for remote collaboration).

5.2.2 Programming by [Remixing] Demonstration

Defining interactive behaviors by first demonstrating and later remixing them is a response to the trade-off between the expressiveness of resulting behaviors and the sophistication of the creation process [223]. This is a simple form of End User Programming (EUP) [224], and as such, faces similar challenges in making the flexibility of computation accessible to non-experts (for an overview, see: [225]). The method used in SketchExpress is hinted from Programming by Demonstration (PbD) [226], which explores how a user’s manual demonstration can specify a program – or interactive behaviors, in our case. To address this challenge, the notion of “remixing” is used, which is commonly used in electronic music: i.e., a DJ chopping, editing, processing, and arranging audio samples to create music. The idea of remixing to facilitate real-time collaboration draws upon the previous work in collaborative improvisation, where musicians can algorithmically remix short musical patterns [227] or musical notation [101].

5.2.3 Summary of Tools for Prototyping Interactive Behaviors

A range of alternatives exists for creating realistic interactive behaviors. Fifteen tools were reviewed to evaluate existing approaches, examining how the tools support a variety of interactive behaviors. Overall, each tool falls in to one of the following three categories.

1. [high programmability - rich expressivity]: these applications (e.g., Kitty, Sketchify, Flinto, Origami) provide methods with which a user can specify relationships and states between objects, equivalent to a programming environment, which yields interactive and dynamic sketches. This class of applications often comes with numerous complex configurations that tend to be time-consuming to learn and understand in order to harness the applications’ full range of expressivity.

2. [preset behaviors for target applications]: these widely used prototype tools (e.g.,

InVision or Adobe Experience Design) provide a limited library of behaviors that a user can choose from for typical common applications (e.g., page transitions, image overlays, and hyperlinks). However, users may struggle both with complex configuration, which is proportional to the number of prepared behaviors, and limitations of the existing preset if the desired behavior is not one of the predefined ones. Hence, this class of solutions cannot handle general applications, such as games or animated illustration.

3. [linear timeline]: these applications (e.g., Atomic.io, Adobe After Effects, K-Sketch) provide a linear timeline editor that is typically available in film-editing software. While this class of applications offers rich expression—as a designer can manipulate elements over time-dimension (frame by frame)—the linearity of the animation limits the dynamics and interactivity of the prototype. Typically, one behavior can be expressed linearly, but a GUI prototype that has a number of behaviors cannot be created using a single timeline-based animation because it can require different outcomes depending on how a user interacts with it. For instance, pawns in a chess game have limited behaviors, but the number of possible states that a chess game can end up in is extremely large.

Most of these applications provide programming-like functionality (or something equivalent) for prototyping interactive behaviors. However, generating expressive animations often comes at the price of learning the tools in depth and attaining expertise in at least one of the required programming concepts. This can be a barrier when utilizing crowd workers to prototype interactive behaviors. Using SketchExpress reduce the effort needed to create interactive behaviors via the demonstrate-and-remix-replay approach. Reducing the effort needed to create animations provides allows requesters to explore interactive behaviors more quickly. Rapid prototyping and iteration is particularly important as the system targets the early stages of the design process, where people often exchange ideas by drawing on a piece of paper (or so-called “napkin sketch”).

5.3 SketchExpress: System Description

SketchExpress’s goal is to let crowd workers help create and power behaviors in interactive GUI prototypes easily and accurately. To recap, the primary challenges are:

- archiving manual demonstrations as reusable and replayable animations;
- allowing for remixing of manually-demonstrated animations to have more precise timing;
- creating interactive behaviors that animate multiple elements simultaneously;

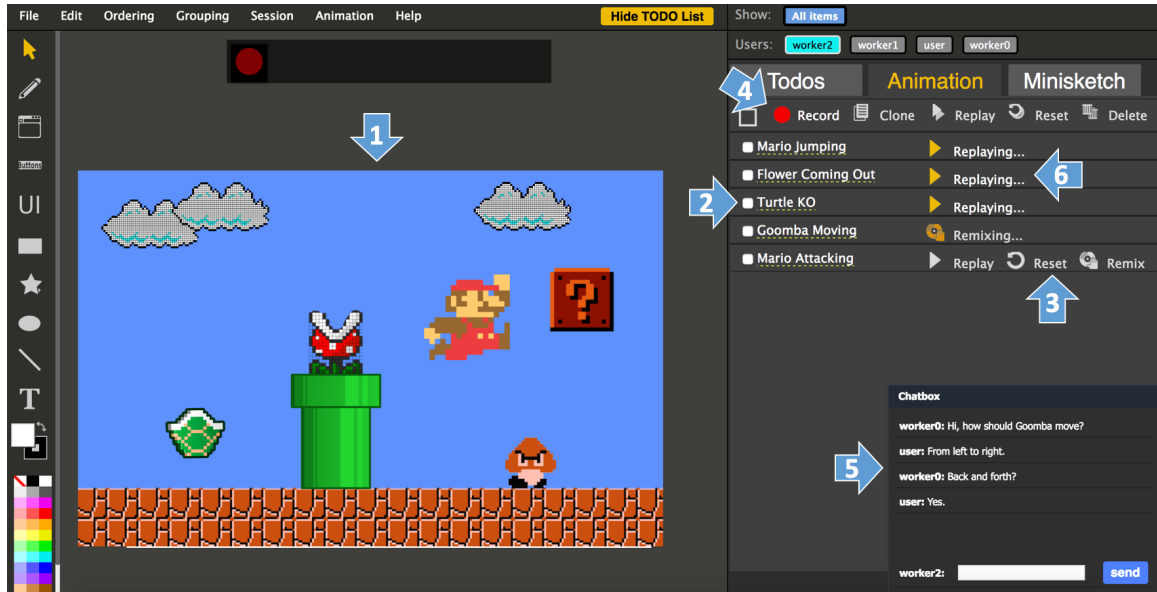


Figure 5.1: SketchExpress allows crowd workers to prototype interactive behaviors. A requester describes aloud how a user interface should behave and crowd workers quickly create complex interactive behaviors. The interface contains the following features for crowd workers to easily create interactive behaviors (animations) as follows: (1) A synchronized canvas that supports simultaneous interactions between a requester and workers. (2) the ability to select and replay multiple animations at once; (3) reset functionality that places elements in the animation back to their initial state (position, color, etc.); (4) recording button to record a worker’s demonstrations; (5) a chat box for helpers to ask clarification questions if needed. (6) labels that show the current state of the animation [replaying/remixing] to prevent multiple workers from concurrently working on the same animation.

- allowing crowd workers to learn to do the tasks above within minutes of first using the system.

The rest of this section introduces the animation functions that crowd workers are able to use to prototype interactive behaviors, describe the implementation of SketchExpress in detail, and discuss how it addresses these challenges.

5.3.1 Platform

SketchExpress is built on top of the existing web-based Apparition system (see [155] for more detail) which provides a shared canvas (modified from SVG Edit, a web-based SVG drawing application [228]) where a requester and a crowd worker can collaborate in real time. A requester verbally describes an interactive behavior via streaming audio channel while sketching on the canvas. The requester's interface is same as the worker's interface except that it has a simplified tool bar with only a free-hand drawing tool (pencil) and a select tool. As the canvas is synchronized in real-time, the requester and the workers see the same content and updates.

5.3.2 Recording Demonstration

SketchExpress provides the ability to record manually demonstrated behaviors, which are stored as a series of time-stamped snapshots of the element that can be later replayed as an animation. To record, workers: 1. press record (Fig.5.1-4), 2. demonstrate the behavior on the canvas, 3. stop recording, and 4. the server post-processes the recorded log to construct a replayable animation. Each recorded animation can be *replayed*, *reset*, and *remixed*. The state of each animation is shared in real time in the side panel to provide awareness (Fig.5.1-1), which helps avoid conflicts in replaying and remixing animations. Pressing the *Replay* button triggers the animation to begin again. Pressing the *Reset* button restores the initial states of the elements used in the animation (Fig.5.1-3).

One key benefit of using the record-and-replay method compared to manual demonstration is that the interactive behaviors then persist as part of the sketch and can be replayed later. While the recording function simply logs all the snapshots of the elements that are changed by the crowd worker, Step 4 categorizes events and generates a series of operations.

There are three supported types of operations: *create*, *change*, and *delete*. The *change* operation can be broken into five sub-categories: *move*, *rotate*, *resize*, *fill-change*, and *stroke-change*. Depending on its type, one operation can have a series of multiple *events* (e.g., *move*) or a single event, typically color changes (e.g., *fill-change*,

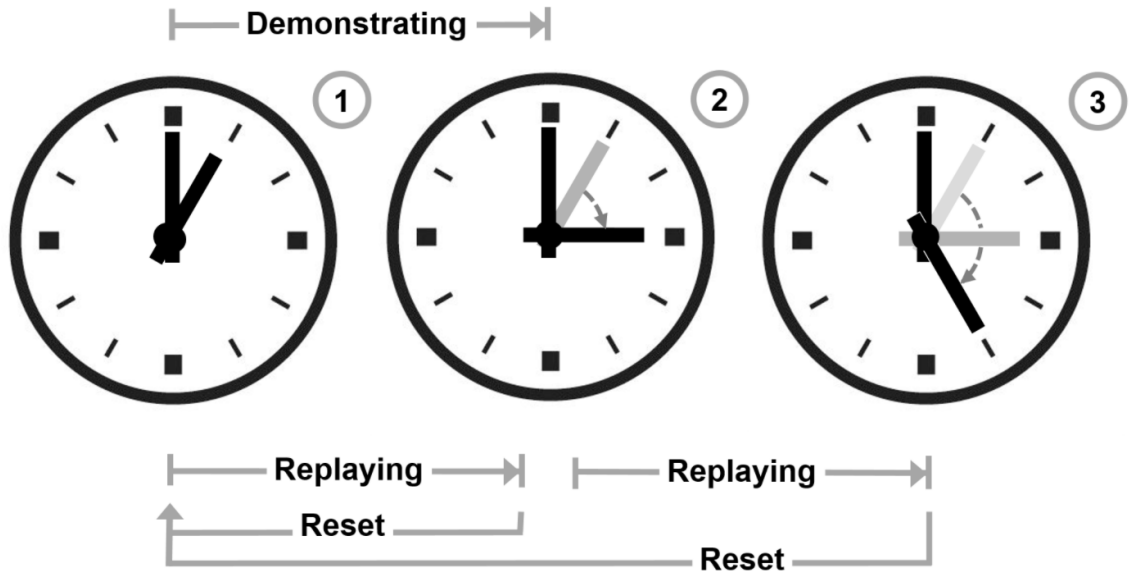


Figure 5.2: (1) Initial State: arrow pointed vertically upwards. (2) After first replay: arrow pointed 60 degrees clockwise. (3) After second replay: arrow pointed 120 degrees clockwise. Reset will restore state (1).

stroke-change). When replaying an animation, each operation is reproduced in real-time as it was demonstrated during recording, including the delays between operations. The operations are listed, showing the type of each operation and the associated timing, in a table in the remix panel (Fig.5.4).

5.3.3 Recording and Replaying By Delta

In the post-processing step (Step 4), two adjacent events are compared to compute the difference (Δ) between two snapshots within an operation. The replayed behavior is thus the relative state difference between the initial state and the ending state ($A' - A$) rather than an absolute frame-for-frame reproduction of the demonstration. There are a few advantages in terms of expressiveness to replaying an animation by delta (as opposed to via a series of snapshots).

First, an element can 'own' a behavior instead of the behavior being reproduced exactly as it was demonstrated. Depending on the current state of the element, each behavior may result in a different animation and yields different outcomes. For example, if an animation is created by recording the rotation of an object by 60 degrees, the resulting animation is not an exact reproduction of the originally recorded animation but instead a rotation by 60 degrees from the element's current position (see the example in Fig.5.2). Second, this

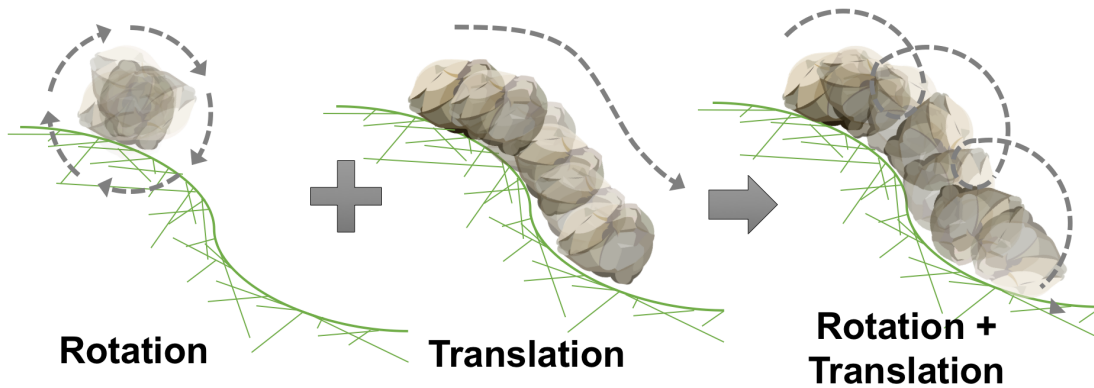


Figure 5.3: SketchExpress facilitates the process of creating complex animations involving multiple transformations on one element. In this example, a worker can create separate *rotate* and *translate* operations, and then replay them together to create the rolling stone animation.

enables multiple transformations on one element. Automated replay of multiple animations enables complex behaviors that cannot be easily demonstrated by manipulating them (see the rolling stone example in Fig.5.3). A worker can replay and combine animations in any order to simulate UI behaviors.

5.3.4 Remixing Animations

SketchExpress provides remixing functions to control the timing of each operation of a recorded animation. While manual demonstration can be spatially expressive, the temporal execution of the demonstration is limited by the time it takes to physically animate the elements. Using remix, workers can adjust the duration of each operation in an animation.

The main interface for remixing is the remix table (Fig.5.4-1). For each operation, there are three options to choose from: *instant*, *skip*, and *real-time* (Fig.5.4-3). *Instant* makes the transition from the operation’s initial state to the final state occur immediately, which is useful when the intermediate operations are not needed in the replay (e.g., for a “teleportation” animation). *Skip* allows one to bypass the recorded operation, which is useful to remove unnecessary actions captured while recording. *Real-time* replays the operation as it was demonstrated with a specified duration that can be stretched or compressed (Fig.5.4-4). *Real-time* exactly reproduces the recorded demonstration if the duration is not modified.

Depending on the type of animation, the initial state of each replay needs to be reset before/after replay and can be looped when it is periodic (e.g., a non-player character patrolling in a game). These options not only automate some of the process, but increase the

kinds of behaviors that SketchExpress can present. Overall, remixing an animation in the temporal dimension is a key function in transforming a demonstration into a precisely timed animation, which not only makes the animation look smoother, but also allows workers to demonstrate behaviors without being concerned with making the initial demonstration perfectly temporally accurate.

One challenge for workers is to associate the contents of the remix table with the animated elements on the canvas. SketchExpress provides multiple visualization techniques for workers to connect entries with canvas elements. First, when recording an animation, the list of operations is generated on-the-fly during demonstration, allowing the worker to immediately associate actions on the canvas with entries that will later be used to remix the animation. Second, whenever an animation is replayed, the entry corresponding to the currently-playing operation is highlighted. The delay row is highlighted if the animation is in between operations. Lastly, whenever a worker places their cursor over one a row in the table, the elements associated with that operation are highlighted (as seen in programming environments that highlight program outcomes associated with code text [229, 230]). In order to clearly visualize “what is remix-able”, SketchExpress uses a consistent format: a yellow-dotted line under options throughout the remix table that can be clicked and remixed.

One of the benefits of the demonstrate-remix-replay approach is that it is easy for non-experts to understand the controls that they are given. The expressiveness afforded by SketchExpress is defined by the multiplication of two orthogonal dimensions (time and space). For spatial dimension, it includes any change that a worker can make in the drawing application, which can be controlled in the *demonstrate* phase. For temporal dimension, there are four types of control actions that can be conducted per operation: *compress*, *stretch*, *skip*, and *instant*. While I could have created remix functions that can modify the spatial data of a demonstrated behavior, the remixing capability is deliberately limited only to the temporal dimension of an operation and delays in between, helping workers more quickly understand the tool’s range of expressiveness. Therefore, if the demonstrated behavior is not spatially correct, workers need to re-record the behavior again, leveraging humans’ fine motor function for the expressiveness given there’s no time pressure. Adding spatial remix functions to correct visual trajectory of animation would have increased the complexity of using the tool and the simple structure of demonstrate-remix helps non-expert crowd workers learn to use the tools by themselves and use them after a brief exploration.

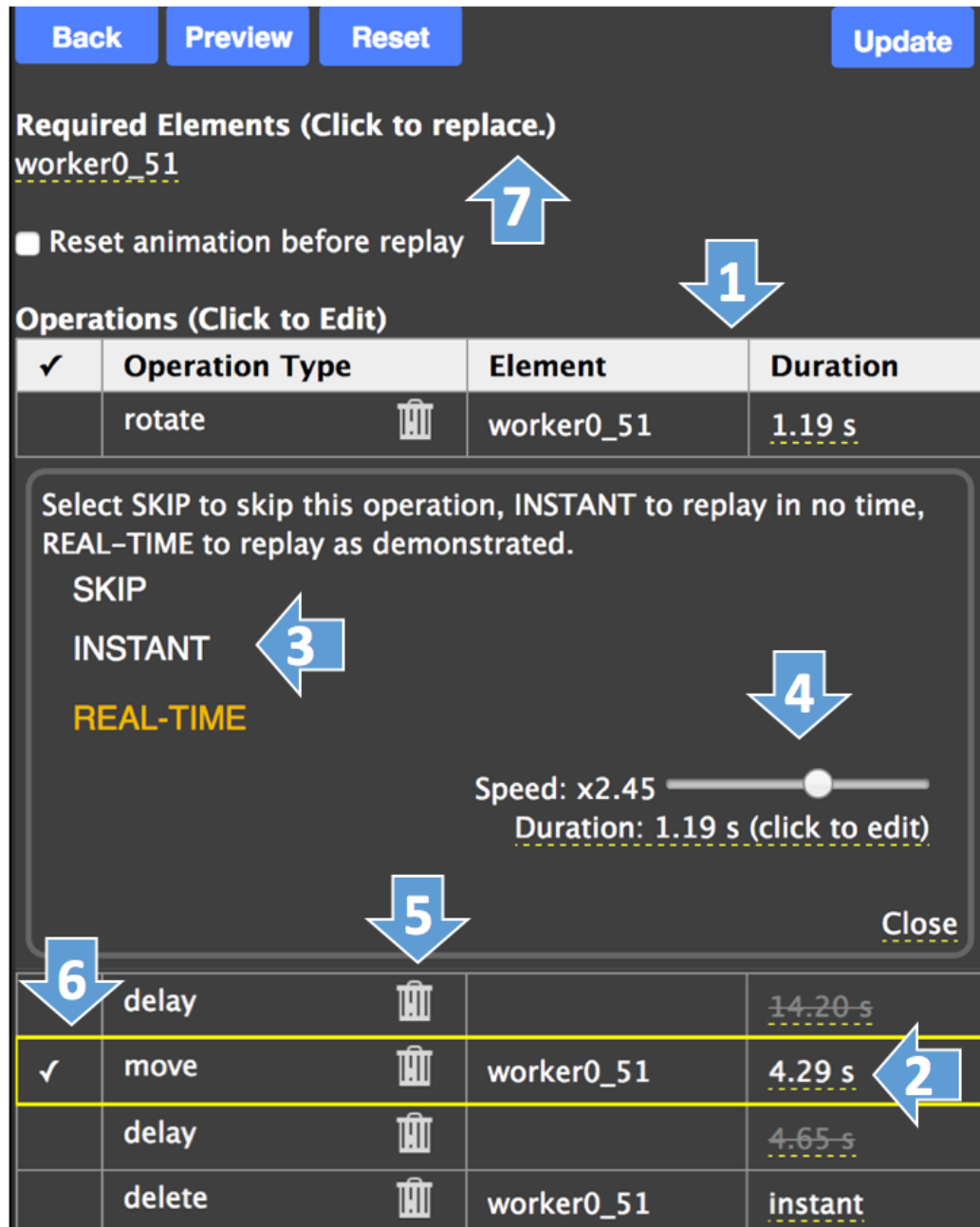


Figure 5.4: Remixing helps workers make more expressive animations. The interface consists of: (1) Operation list: provides workers with a discrete view of an animation as a series of operations. (2) Operation duration: if clicked, a container of remix functions is expanded. (3) Replay options: you can choose for each element if it will be skipped (not displayed), if it will appear instantly, or if it will appear in real-time. (4) Slider and input: modify the operation’s speed and duration. (5) Trash icon: skip the operation or delay. (6) Check mark and highlight border: indicate the current operation in preview. (7) Element replacement: reuses the animation for one element as the animation for another one.

5.3.5 Animation as a First Class Object

SketchExpress provides features that treat animations as independent from the elements on the canvas – akin to a first-class programming object. For example, a worker can “clone” an animation and switch the element that is used in the animation so a certain behavior can be applied to different elements (e.g., letting us apply our example turtle knock-out behavior to other enemy characters). A worker can replace existing elements of an animation in the “Required Elements” and “Created Elements” list below the buttons in the remix mode (Fig.5.4-7). Once switched, any operation that was associated with the original element works for the new one. To avoid orphaned animations, when a user deletes an element associated with an animation, SketchExpress alerts a worker with a list of the affected animations. Cloning an animation can be used to create multiple remixed versions of one demonstrated behavior. Finally, animations can be imported/exported across sessions by archiving them *json* content. Treating animations as first-class objects lets workers easily compose new animations.

5.4 Laboratory User Evaluation

5.4.1 User Study - UI Tasks

To verify SketchExpress’ ability to help crowd workers prototype interactive behaviors in various UIs, a user study is conducted. In the study, crowd workers were given behavior descriptions and asked to collectively create them using our interface. Five common interfaces were selected to incorporate complex interactive behaviors (difficult to manually demonstrate) from various domains (from mobile to game design). The study controlled for variation in natural language descriptions by having one of authors read from a script describing the tasks across the sessions. Since the goal of this work is to confirm if crowd workers can create behaviors easily and accurately, the chance that confusion would arise from variations in either verbal communication or the description of task content was controlled by having the fixed script. The script is carefully generated to reflect the target use cases by transcribing a non-designer, verbally describing the interactive behaviors in the tasks. As this study focuses on system feature effects on the interactive behaviors, not the static parts of the sketch, a graphical user interface is given to crowd workers and the interface has all the elements necessary for a worker to demonstrate the behaviors that will be requested. Crowd workers are instructed to create interactive behaviors for each task, resulting in a total of nine interactive behaviors across five sketches. Each task (T) and interactive behavior (IB) is described to workers as follows:

- Task 1 (**T1**): Super Mario Game – on the ground, Super Mario jumps to defeat a turtle (a.k.a. Koopa) and an enemy mushroom (a.k.a. Goomba)
 - IB1: Super Mario jumping forward
 - IB2: Turtle Knock out gesture
 - IB3: Mushroom Knock out gesture
- Task 2 (**T2**): Traffic lights Demonstration
 - IB4: Traffic light changing color from green to yellow to red with a two second delay between each change
- Task 3 (**T3**): To-do List Application
 - IB5: Crossing off an item (the 1st item in the list) by showing a check mark in a check box and a strike-through the text simultaneously and instantly
 - IB6: Crossing off an item (the 2nd item in the list)
- Task 4 (**T4**): A cannon-firing game
 - IB7: A cannonball from the pile of cannonballs is loaded into a cannon barrel, at which point the cannon shoots it out to destroy an enemy character
 - IB8: Same as IB7 for another cannonball and the second enemy character
- Task 5 (**T5**): Unlock screen
 - IB9: A user “swipes to unlock“ a smartphone screen

These tasks focus on remixing animations rather than the reusability, for which benefits may emerge over time. For example, making IB3 could have benefited from re-using IB2.

5.4.2 Participants

18 unique crowd workers from Mechanical Turk were recruited. They have never used SketchExpress before, are in the U.S., and have an approval rate of over 70%. All workers who applied for the work were asked four binary questions to see if they were eligible to complete our user study: 1) if they can listen to verbal instructions through audio streaming, 2) if they are familiar with at least one common creative application (Microsoft PowerPoint/Microsoft Point/Google Draw/Adobe Photoshop), 3) if they are using a specific web browser with which SketchExpress has been developed and rigorously tested, and

4) if they have sufficient time to complete the entire study (which ranged from 30 to 60 minutes). If one or more of the answers were negative, they were paid only for filling out the pre-screening survey (a flat rate of \$0.30). If they were eligible for the study, they were directed to a tutorial video made for the specific condition they were in (max 4 minutes).

Once they finished watching the video, workers were routed from a retainer pool to the task interface in advance to ensure they are available when needed. Once at the task page, workers were on standby for the span of multiple requests (from **IB1** to **IB9**) for a single session. Workers were paid a base rate of \$10.20 per hour. At the beginning of each session, all participants were given a brief introduction to the experiment and were asked to familiarize themselves with the application by exploring what was covered in the tutorial video until they felt comfortable using the tool (warm-up time). During the warm-up time, workers were not given specific instructions unless they asked for clarification. At the end of the warm-up time, the requester checked if workers knew how to use the set of functions that were required to solve the tasks, which was included in our measure of warm-up time duration.

5.4.3 Experimental Design

Our study had three experimental conditions: **(C1)** the control condition, which used manual demonstration only (recreating [155]), **(C2)** the demo-and-replay condition where the application let workers record and replay animations but had no remix function, **(C3)** and the demo-remix-replay condition (the SketchExpress condition) that contained all proposed system features. Comparing the control condition (**(C1)**) with the other two (**(C2)/(C3)**) allowed us to understand the effectiveness of the demonstrate-remix-and-replay approach. The intermediate condition of **(C2)** is added in order to account for the potential improvement (or detriment) in completion time or the accuracy. The control condition (**(C1)**) reflects the original model used in the previous work, in which crowd workers listen and respond to demonstrate the described behaviors [155].

Each crowd worker was randomly assigned to one of the three experimental conditions (a between-subjects design) and each was asked to complete five tasks. Though the order in which the five tasks were presented was randomized, the order of the interactive behaviors within each task was fixed in the order presented above. Three workers left their session without completing all the tasks. This led us to recruit more workers so each condition was completed by the same number of workers.

More specifically, tasks were conducted in the following order:

First Demo. A requester described an interactive behavior verbally and asked crowd workers

to demonstrate (C1) or create an animation for it (C2, C3). Once the crowd believed they were done with the demonstration, they gave a “done” signal to the requester by changing the color of a circle (from red to green). If a worker asked any clarification question, the requester repeated the description one more time.

Second Demo. Once all interactive behaviors in a task were completed, the requester asked workers to demonstrate each behavior once more. Workers could use recorded (and remixed) animations in C2 and C3, while in C1 workers had to manually demonstrate the behavior each time. I asked workers to demonstrate behaviors twice in order to validate the benefits of reduced time for replayable interactive behaviors with a button click compared to manual demonstration.

5.4.4 Performance Measures

For each interactive behavior, I measured the accuracy by calculating precision and recall. Precision and recall indicate the overall quality of created animations according to the verbal description given to the workers. This was done based on scoring rubrics that I created to evaluate the crowd workers’ demonstrations. The rubrics were created based on the verbal description that I provided to the crowd workers¹. The annotators countered potential bias using well-defined yes/no questions, not subjective ones. Some of the examples include:

- *Do the operations happen in the correct order? (Task 1,2,4,5)*
- *Does Super Mario move forward? (Task 1)*
- *Is there only one light (at least, and at most) on at any point in time? (Task 2).*

To calculate precision and recall, annotators counted the number of rubrics that were satisfied (True Positive), the number of rubric entries that workers missed (False Negative), and the number of unnecessary actions in the animation (False Positive). In addition, we manually annotated animations’ start and end times, as well as other relevant events, such as replay, record, remix, requests for clarification, and reset (if available in the condition). These timestamps were used to calculate the average time spent completing each request. We manually annotated both the time and accuracy for three videos and assessed the consistency of these quantitative measurements by calculating intraclass correlations (ICC). The annotators had perfect agreement on precision and recall (ICC: 1.00) and nearly perfect agreement on time annotation (ICC: 0.99). We also annotated how long each participant spent getting familiar with the tool (warm-up).

¹Our rubrics are available for download here:
<https://sketchexpress.github.io/rubrics.html>

5.5 Result: a Sketch That Behaves

This section presents SketchExpress' quality and latency performance in the context of our two requests (First Demo and Second Demo). The statistical significance was examined with pairwise 2-tail *t-test* between three pairs of conditions (resulting in a total of three t-tests per metric).

5.5.1 Improving Quality

Figure 5.5 shows that SketchExpress significantly improves animation quality, resulting in 90.0% ($\sigma = 11.2\%$) overall recall for our system condition (C3) – a 27.3% improvement compared to the control condition (C1, 62.7%, $\sigma = 19.8\%$ $p < 0.001$). However, the recall in the demo-replay function without remix (C2) was not significantly different from the control condition (C1). This indicates that when the requested behaviors are complex, the addition of a remix function is critical for improving recall. On the other hand, when creating the simplest behavior, (**IB1**, which needed only one operation), there was not a significant difference between C1 and C3. Observationally, this is because the simpler behavior could be manually demonstrated accurately, thus there was not much room for improvement using remix.

There were significant improvements in precision across all conditions. Having replay function leads to a gain of 7.8% in precision, and the effect was statistically significant ($p < 0.001$). Precision in the system condition (C3) is almost perfect (99.2%, $\sigma = 2.9\%$), which is not surprising given that workers could choose to *skip* unnecessary operations in the remix mode, which was not available in (C1) and (C2). Remixing results in a 12.3% increase in precision when comparing (C3) to the control condition (C1, $\sigma = 20.7\%$, $p < 0.001$).

When replay functionality was available, the adoption rate was very high: when a requester asked for the behavior to be demonstrated the second time, it is observed that *all* participants with access to replay (C2, C3) chose to use it for the animation they already created, instead of performing a new demonstration. The resulting sketch in the control condition (C1) was a static drawing of a graphical user interface that does not contain the interactive behaviors during the session, the sketches in (C2) and (C3) included behaviors that can reproduce the behaviors that were described in future sessions.

5.5.2 Improving Long-Term Latency

The First Demo result shows that, as anticipated, it took significantly more time to demonstrate *and* remix a demonstration (C3, 174.5s, $\sigma = 114.4s$) on average than it did to just

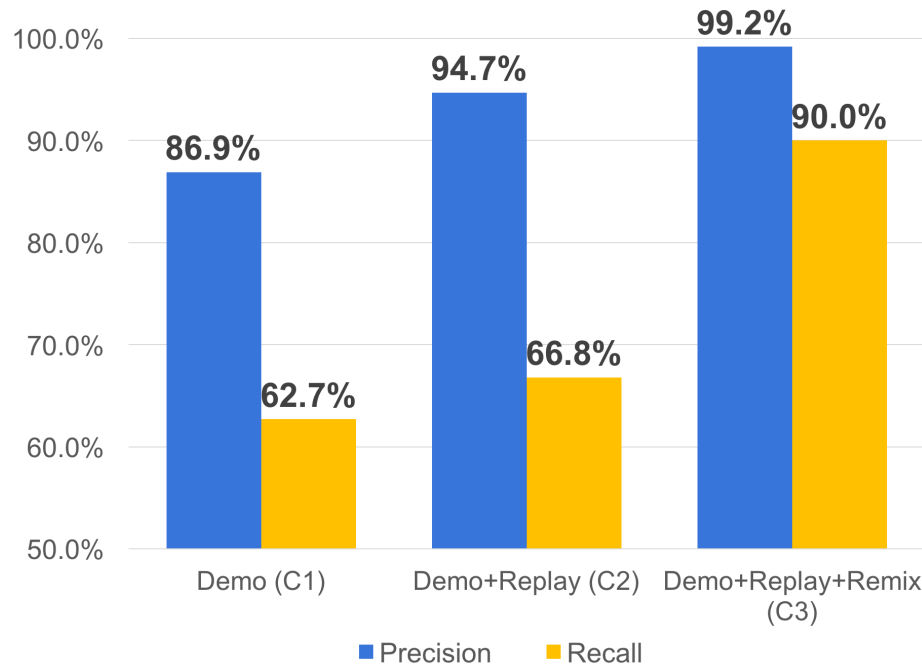


Figure 5.5: Though there is not a significant difference in recall between (C1) and (C2), recall is significantly higher in (C3) compared to (C1) and (C2) (both $p < .001$). There are significant increases in precision from C1 to C2 ($p < .001$) and from C2 to C3 ($p < .001$).

perform the demonstration itself (C1, 39.4s, $\sigma = 46.7s$, $p < 0.001$). Even just recording the demonstration and replaying it to review added time compared to the control condition (C2, 78.3s, $\sigma = 60.6s$, $p < 0.001$). In addition, workers spent more time warming-up to the demo-remix-replay condition (C3, 10.2 min, $\sigma = 1.99m$) on average than they did in the control condition (C1, 5.44 mins, $\sigma = 3.49m$, $p < 0.05$). There was not a significant difference in warm-up time between the remaining two conditions (C2, 6.9 mins, $\sigma = 3.67m$).

While the initial creation of higher quality animations takes more time, once the behavior is recorded and remixed, it allows workers to respond very quickly to requests by replaying existing behaviors. The average time it takes to perform an animation the second time is 35.6s, 19.1s, and 12.4s in (C1), (C2), and (C3), respectively. Importantly, the demonstration speed in (C3) was significantly faster than it is in the control condition (C1, $p < 0.05$). The main source of this difference comes from the methods used to restore the initial state, which is depicted by the green portion of the graph in Fig.5.6. In the control condition (C1), a participant needed to manually restore the state to re-demonstrate a behavior on the canvas, while in the other two conditions, participants reset the canvas using the ‘Reset’ button. This is especially effective in the demo-remix-replay condition (C3) as workers frequently reset the state while they are remixing an animation. This result has implications for interface

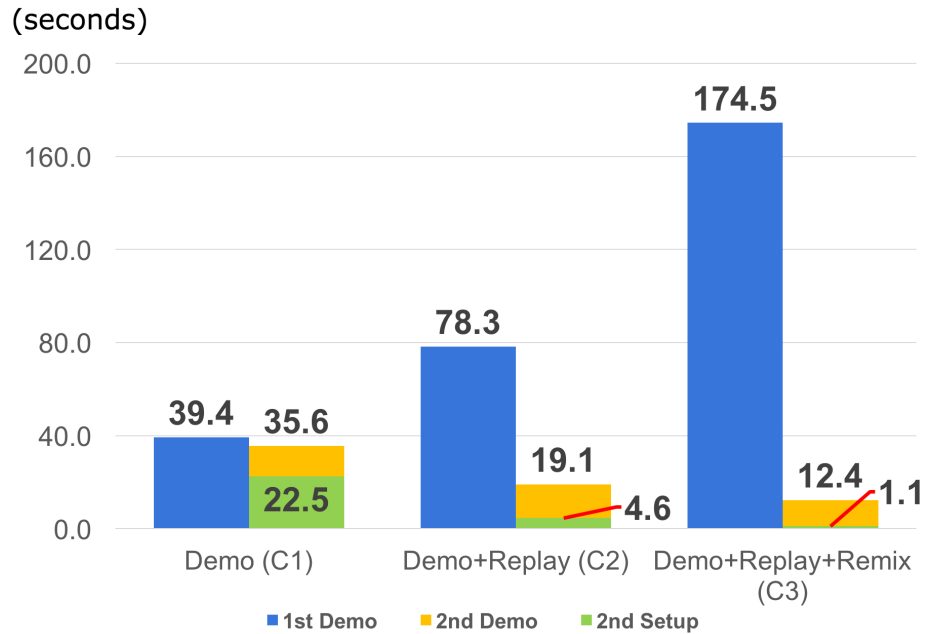


Figure 5.6: Latency of the 1st demo(blue) and the 2nd demo(yellow); The time it takes to demo-remix-replay an animation is longer than the other two conditions, but once an animation is created (the 2nd demo), a worker can respond quickly by replaying the animation. This is because the amount of time needed to respond to the demonstration request is the time it takes to restore the initial state needed to reproduce the requester behavior (the portion of the green bar in the yellow one).

prototypes that utilize the same animation multiple times. Having both remix and replay functions potentially makes the creation of prototypes that use the same complex behavior multiple times even more efficient.

5.5.3 Task Engagement

One interesting observation was that crowd workers constantly tried to refine the animation in two conditions with demo-remix-replay (C2, C3), indicating high task engagement. The number of trials, the number of requests to clarify the user request, and the number of replays of the intermediate results is higher in the demo-remix-replay condition than in the other two (if available). Several crowd workers “rehearsed” the demonstration before recording. Some crowd workers spent additional time re-demonstrating and remixing the animation in (C3) even after generating a sufficiently accurate animation. In these cases, it was not clear why the workers kept trying to re-demonstrate and refine the behavior as the animation generated was already good-enough, but some workers spent an excessively long time on the task (e.g., maximum time: 13.3 minutes for one animation), though previously

work would categorize this type of worker as an “eager beaver” [111]. This indicates that the need to interrupt the excessive improvement to avoid wasting effort, but the natural desire to improve on the reusable components is promising.

In general, workers appear to be actively engaged with the tasks. Though there was no formal survey, five crowd workers voluntarily provided positive feedback about the task in the chatbox. To name a few: *“This is fun”*, *“It was a great study”*, *“Wow, this is great. I become an animator.”*, *“How can I download animation on my PC?”* This is promising because if crowd workers find these kinds of tasks more engaging than other tasks available on MTurk, it will be easier to re-recruit participants who have already used our interface, allowing workers to gain expertise in the task over time.

5.6 Conclusion

This chapter presents the design, implementation, and evaluation of SketchExpress, a system that enables non-expert crowd workers to quickly and easily create replayable animations that persist in electronic sketches. This, in turn, helps requesters more effectively prototype interactive UI behaviors. Crowd “Wizards-of-Oz” can quickly and accurately create replayable animations in minutes with a 27% improvement in recall.

Future work aims to explore how the system can learn from different instances of behavior created by multiple crowd workers, as well as edits made in the refinement steps to generalize a class of animation into an interactive behavior. Ideally, the system can automatically vary the animated behavior by itself depending on the different system states and settings. For example, Super Mario may jump differently if there is a brick wall in front of him compared to if the path ahead is clear. In the future, we plan to use machine learning to learn the structure of interactive behaviors and analyze the crowd’s demonstrations based on requesters’ verbal descriptions of behaviors. Eventually, we hope to develop a computational system that can help automate the creation process through the use of both human and machine intelligence.

CHAPTER 6

Crowd in C: Audience Participation in Music Performance⁵

6.1 Introduction

In this chapter, we introduce *Crowd in C*, a computer music piece designed for large-scale audience participation at a music concert using their smartphones. It has been a long-standing endeavor to create musical performances in which the audience can easily participate. In particular, mobile smartphones have the highly desirable characteristic of already being in the possession of the audience members while offering networking and rich sensor capabilities. This work draws upon a long-standing research and performance traditions of developing groupware that enable distributed musical performances. The key component in this work is the ways in which to involve non-expert audience members in music performance. Challenges of the audience participation in music concert is as follows:

- the system needs to be immediately accessible to non-expert audience members who do not have any musical background.
- the artifact created (music, in this case), solely relies on the participation.
- the system needs to allow musicians (creators) to orchestrate the audience members to perform a musical piece without interrupting their participation.

Crowd in C will address these challenges by providing a musical instrument on a smartphone that facilitate social interaction among audience members and building a mobile ad hoc network(MANET) with which a musician can control the music.

⁵Portions of this chapter appear in [62, 69, 231] with the permission of Antonio Deusany De Carvalho Junior, the lead author of [69].

6.2 Motivation: Crowd Playing and Mingling in C

As the name of the system hints, the crowd (audience) plays the musical instrument in C Major scale. This is directly inspired from the piece *In C* by Terry Riley [232]. In the piece, musicians (with various instruments) were guided to play pre-composed melodic fragments in sequence for the random number of time. As the number of playing one fragment are left to each musician's decision, the collective outcome of the ensemble creates heterophonic texture with chance largely in C chord. Similarly, in *Crowd in C*, each audience member will play a series of short snippets composed by audience members including oneself. The interface provided will first guide a participant to compose a short "tune" that has five musical notes in C major scale. Once the participant finishes the composition, he or she can browse what other audience members composed and play the tunes made by the participants. Therefore, it is very similar to Terry Riley's *In C* in a way that one decide on how long to play a tune. The difference is that there is no pre-composed fragments but each audience member will contribute to build the piece by submitting a short melodic tunes. In this way, participants will have their own tunes and a chord scale becomes the common ground upon which all audience play.

In addition, there is a separate musician performing the piece on stage at the same time with the audience members. The role of the musician is a meta-performer who can control the chord scale in which the audience members are playing. For example, the meta performer can change the instrument tuned in C major scale to a different chord scale (e.g. C Minor, Pentatonic Scale) on the fly. In the meantime this performer cannot generate sound at all on his/her end. Rather, the performer only controls the harmonic flow of the piece generated by the crowd. The interplay between the musician and audience members assure that each audience members will play individual patterns while a musician can progress the piece by changing chord scales. This performer-audience pairing model is coming from the previous work of *echobo* [61] where audience members plays a simplified key instrument on smartphones with the chord progression determined by a performer on stage and synchronized over a mobile network. The technical detail of how the performer changes the scale will be discussed in the later section 6.4.2.

6.3 Collective Creation of Music

6.3.1 Loop-based Instrument

The web-based musical instrument we developed for the piece contains a simple interface that can loop a five-notes melody in a specified scale (C major scale in the beginning). There are five circular notes (or "note dots") that are connected by lines. And there is a circular play head (or "the play dot") in yellow which travels five red (or green) note dots, which triggers a tone whenever it reaches a note dot. The play dot moves at a constant speed so that a melodic pattern (or "Tune") will be looped consistently. This assures that the instrument will generate sound without any user involvement as long as the user turns up the volume and stays on the page so that the musician need not worry about being too sparse or silent due to low participation. The expressive range of the instrument depends on where a player places note dots on screen. First, the vertical position of note dots determines the pitch of the notes. The interface visualizes pitch difference with alternating white and gray divisions in the background. Secondly, the horizontal position of a note dot determines the timbre of the tone; the note dots placed in the leftmost side of the screen will generate pure sine tones while the note dots on the other end will play a tone that combines various oscillators (sine, sawtooth, square and triangle waves with different detune parameters).

Sound synthesis of the instrument is entirely done using Web Audio API oscillators. The instrument implements a javascript object for each tone (or "voice"). Each time the play dot reaches a note dot, the program creates a voice instance that contains a set of oscillators. The voice instance includes a javascript object that contains a Gain node and implements ADSR envelope. The interval between two consecutive notes will be determined by the length of a line in between and the duration of each note is proportional to the interval. A tune can be archived with the position data of five note dots in order and the archived data can be later shared with other audience members to reproduce the tune in other devices. Since each audience member can use a mobile phone with a different form factor (or screen resolution, more specifically), all position data of note dots is scaled to the range of [0,1].

Note that the duration of one's tune can be arbitrary long and will not be exactly the same with any other tune of other audience members. We embrace that asynchronicity among ensemble members and leave the temporal expressivity of a tune up to each player. It is similar to the original version of Terry Riley's *In C* where there was no pulse. We find that having the synchronized global pulse and quantized beats will give audience a different experience and achieve a different style of music, which is left for a future work.

We wanted to design the instrument extremely accessible for audience to pick up in a few seconds as well as good-enough for a participant to be musically expressive. In a

fear that audience members may not sustain the interest in playing the instrument, one may criticize that not only the musical affordance of the instrument is limited but also mapping of musical properties are interdependent. Indeed, the mixed use of note dots location for multiple parameters (timbre, pitch and time) constrains the expressive space of the instrument. For example, one cannot play two consecutive notes of same timbre and same pitch with a long interval. While we could have made an effort to build a musical instrument that achieves *low entry and no ceiling* [233], we take a different approach to encourage the participation. We find it acceptable to develop a constrained musical instrument [234] in a hope that participants will discover diversity of playing style via social interaction with other audience members. Therefore, we did not think profoundly to create a nice interface and mapping, especially for this disposable instrument that will be used less than 10 minutes. The greater detail will be discussed in the following section 6.3.2.

6.3.2 Social Interaction with Online Dating Metaphor

As discussed earlier, the musical instrument provided to audience members is limited; it can only loop a five notes with different pitch choices and timbre variation. In turn, once the user finishes the composition of a short snippet, the user is able to browse other people's composition. This is coming from the idea of online-dating website (such as Tinder¹) where a user create a personal profile and then browse other members profiles that include pictures and written descriptions about themselves. Similarly, the networked instrument creates a temporary social network that will last until the end of the performance where a short tune is used as a musical profile. Lastly, the collection of each tune composed by individuals serve as musical phrases that Riley's *In C* is composed of. A musician can play as much as s/he wants and move on to the next phrase. The difference here is that the number of musical phrases is same as the number of participants and each participant can change the short composition on the fly.

Allowing participants to browse other tunes is expected to motivate people to play the instrument in various ways. First, it gives a reason to compose the tune to express oneself, to attract more people and to find a (musical) match among the participants. This resembles a self-presentation strategy in online dating sites where participants post photographs and a written description that represent themselves well. Secondly, browsing the tunes inspire participants to discover new styles to play the instrument. For example, suppose one created a tune with ascending tones in C major scale (like in the left picture of Figure 6.1) and then later discover a tune that uses note dots to visually draw a certain object (like in the right

¹www.gotinder.com

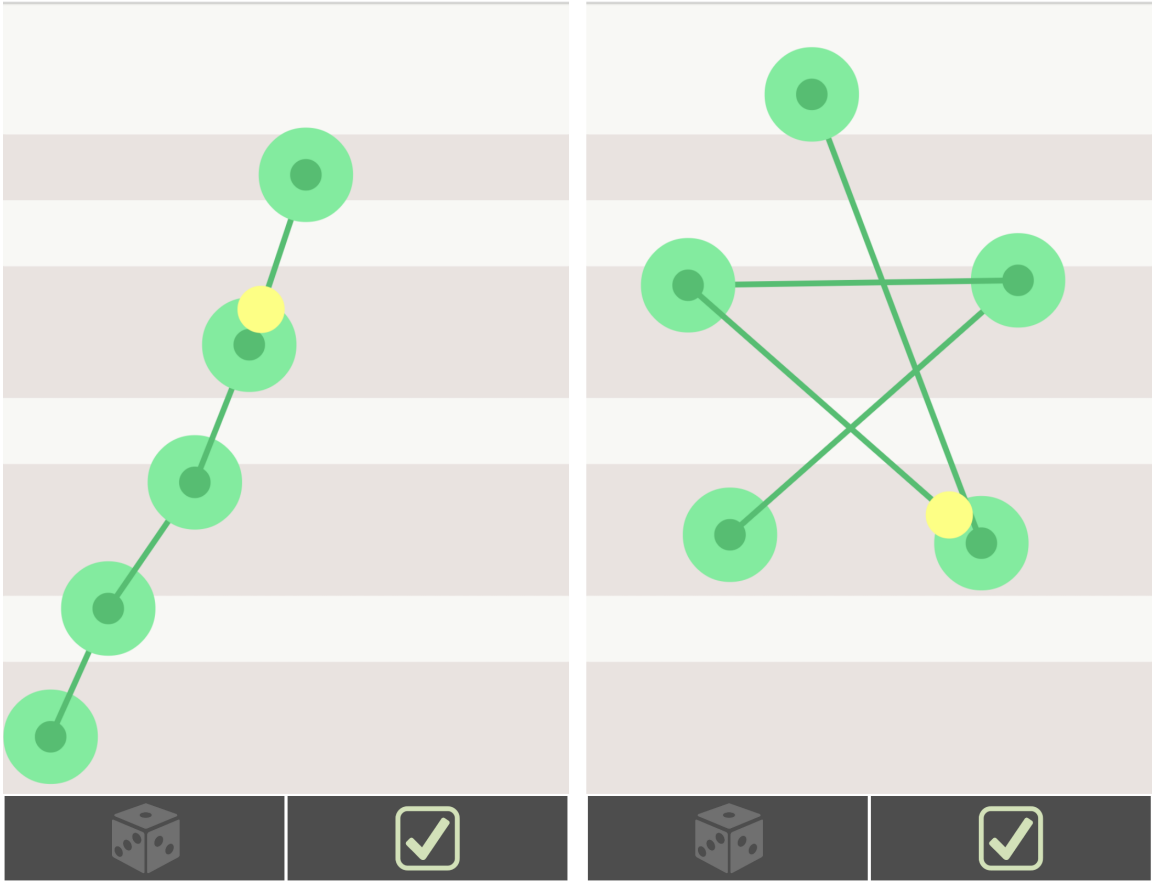


Figure 6.1: Screen-shots of Audience Interface. Playing Ascending Tones (left) and Drawing a Star (right).

picture of Figure 6.1). Later, one may find another tune that have five note dots in one place as close as possible so that it will create a very dense rhythmic pattern with short intervals between notes. Lastly, we bring the joy of playing together. In MINGLE mode, which will be discussed below, one can play his or her own tune with another tune. When two tunes are looped in a same screen, a user is allowed to modify his or her own tune to musically match the tune of the other. One can try to synchronize two tunes to play polyphonic sound, try to alternate notes of two tunes to make a rhythmically dense pattern, or try to make a dynamic pattern by placing dots in temporally, timbrally, or harmonically contrasting ways. It is a metaphor for the situation where two people in online-dating website start conversation, meet off-line and explore the possibility of being a match. We are planning to analyze the interaction of the participants to investigate whether this socially connected ensemble actually inspired each other. While there are many different levels of interaction in online-dating websites, we borrowed the simplest model from a popular online dating application, Tinder, where one can browse the profiles, press like button, start to chat when

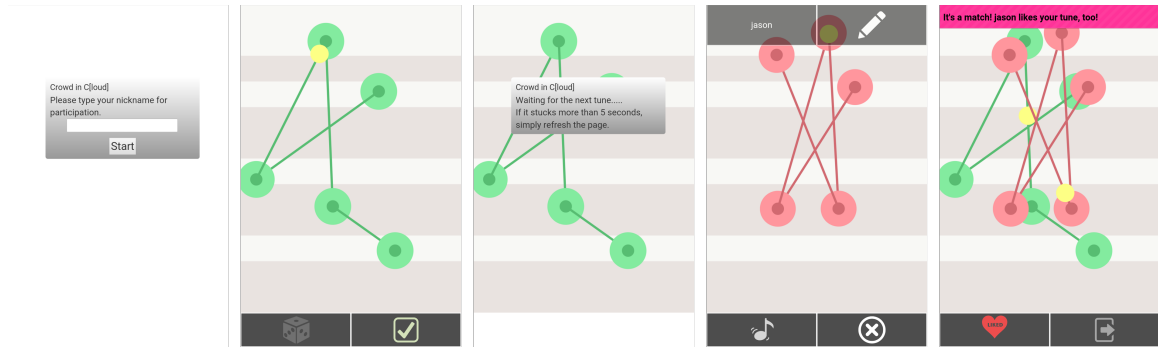


Figure 6.2: Screenshots of Audience Interface in Five States. From left to right: NAME, EDIT, WAIT, CHECK, MINGLE states

it's a match. There are total five different states in which the musical instrument can be NAME, EDIT, WAIT, CHECK, MINGLE. Each state is used to design different interfaces and determine what other states a user can reach from and go to. Five states the instrument are described below.

- **NAME** : When an audience member first visit the link provided <http://bit.ly/crowdinc>, one will be prompted to type a unique screen name that will be used throughout the performance (Figure 6.2-1(the leftmost)). Once the participant submit a screen name, the web page will be redirected to EDIT state.
- **EDIT** : A participant composes a tune in this state by drag and drop the note dots. The play dots will continue while editing so one can hear the current tune (Figure 6.2-2).
- **WAIT** : This is a transient state where the instrument is waiting for a message from the cloud service after a request for data. The incoming message contains data for other member's tune (Figure 6.2-3).
- **CHECK** : This is a state where a participant can browse a tune of the other member (Figure 6.2-4).
- **MINGLE** : This is a state where a participant can play two tunes at the same time (Figure 6.2-5(rightmost)). The note dots in green are the tune composed by the user and the note dots in red are the tune composed by another audience member. In this mode, one can freely move green note dots to make two tunes sound differently and explore a new musical pattern with the combination. In the mean time, the red dots cannot be modified.

The interface is designed to notify social interaction by broadcasting messages. For example, when a participant named John "likes" Jane's musical profile by pressing heart

Crowd in C[loud] : <http://bit.ly/crowdinc>

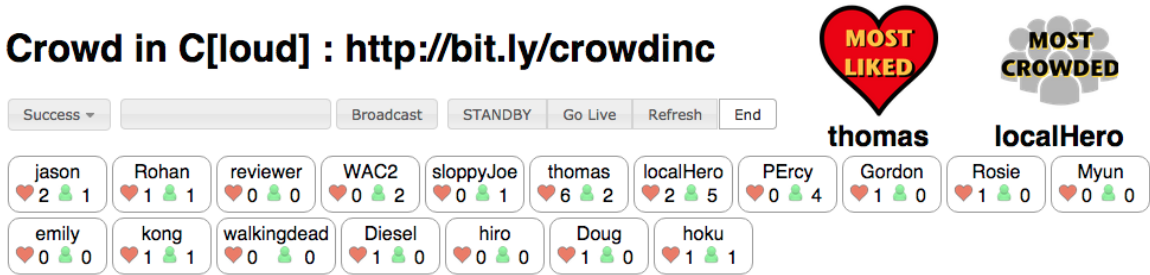


Figure 6.3: a Screen-shot of Performer Interface

shaped button in MINGLE mode, Jane will receive a message saying “John likes your tune!”. Later, Jane browses more tunes and happens to like John’s pattern, he will receive a message saying “It’s a match! Jane liked you back!” in the top banner (Figure 6.2-5(rightmost)).

On the other hand, the performer’s interface (Figure 6.3), which is also a web page, is used to display the list of screen names that are currently participating at the performance. The performer program calculates the number of likes received and the number of participants playing the pattern at the moment under each individual screen name. This interface works like a score-board when projected on screen at the concert hall. On the top right corner, it shows the screen name of the participant whose tune is most liked and the screen name of the participant whose tune is most played at the moment (named “most crowded”). This projection helps audience members realize that the nature of the participation is social and it also helps non participating audience engage with the piece by looking at how their friends and families are doing.

6.4 Crowd Musicking in Cloud

The performer interface and the audience interface are available in the following links.

- Performer : <http://crowdinc.github.io/performer.html>
- Audience : <http://crowdinc.github.io>

To try the demo, press the Go Live button in the performer interface and use multiple devices to play in the audience interface. For now, there can be at most one person who uses the performer interface

6.4.1 Mobile Ad Hoc Network using Cloud Service

As mentioned earlier, we utilized a cloud service to exchange data among audience members and to orchestrate chord scale of the crowd. The performer interface and the audience

interface are two static web pages hosted in a university web server. Once both web pages are downloaded to a device, there is no dynamic interaction between the device and the web server. The performer interface runs on a laptop and the audience interface typically runs on a smartphone of a participant. The performer interface maintain relevant data in local javascript data structure regarding all the participants' data (tunes) and all the information needed to display the scoreboard. Although the performer interface is a web page running on a local machine, it acts as a server in the traditional sense. The only difference is that the server(a performer's laptop) and the clients(audience's smartphones) communicate via a cloud service with minimal network configuration which is already hard-coded inside the javascript file.

We chose PubNub cloud service² after comparison of many cloud services. PubNub is chosen because it provides better bandwidth³ and reliability⁴ While we used a free plan from the PubNub for the development and the rehearsal, we purchased the cheapest paid plan to obtain a dedicated key that will allow more than one hundred participants at the session for the actual performance.

PubNub follows pub-sub paradigm for data communication in javascript. Any number of javascript web application using a same application key can publish (or send) messages to certain channels or subscribe (or listen) to one or more channels. There are three types of channels used in the application; **performer**, **audience**, and **<screenname>**. **performer** is a channel that only the performer program listens to and it is used when audience programs make a request to retrieve certain data such as a tune object. **<screenname>** is given to each participant with the screen name and it is used to transfer data from performer to each individual, which is the response to the request mentioned previously. For example, a participant, whose screen name is **rohan**, is automatically subscribed to the channel named **rohan**. If **rohan** finishes editing a tune, pressing the submit button (that looks like a checked box in Figure 6.2-2) will make the audience application publish a message with the tune object to the **performer** channel. And then, upon the receipt of the tune data from **rohan**, the performer application will publish a message to **rohan** channel with the tune of the first participant in line so that **rohan's** smartphone can render a tune. This structure configures a network where all participants communicate to each other through the performer's application; the clients have one channel to submit data and request for data, and the performer can reach individual audience member through a channel created with the

²PubNub Cloud Service: <http://www.pubnub.com>

³For example, for free plan, PubNub accepts messages of 32KB while Pusher limits the messages upto 10KB.

⁴PubNub limits the quantity of messages sent per month but the limits are never throttled even on a free plan, so one can consider that we can use the full service capacity during one single performance.

unique screen name.

Lastly, all clients subscribe to the audience channel and it is used to broadcast a message or to change the chord scale of participants instruments. For example, a performer can type any message in the performer interface using a text input form and the broadcast button on the top left corner (See Figure 6.3. The typed message will appear on the top banner for a few seconds. This can be used to communicate with audience to communicate with the audience for various purposes (e.g. “The performance will start soon!”, “Look at the projection screen and see who’s most liked”) The more use of audience channel will be discussed later in the following section 6.4.2

6.4.2 Orchestrating the Crowd through Live Coding

As mentioned earlier, the role of performer is to change the chord scale in which audience members play. Without chord scale change, the aggregated sound will be a collection of random notes played in C and will sound somewhat random at some point. Changing the chord scale in an audience application is pre-written as a javascript function and the performer can send signal to audience channel to call the function to make changes in the entire crowd. While there are many ways to signal the function call in audience members’ devices (buttons, knobs, sliders, keys), we chose to live code on the javascript console of the web browser. A performer can type the following line on the console to change the chord scale of the whole crowd.

```
publishMessage("audience", type:"scale", baseNote:60, scale:[0, 3, 7, 12]);
```

`publishMessage` function is written to broadcast a message with a javascript object to a specified channel (`audience` in this case). `type` indicates that the message is to change the scale and the parsing function in the audience application expects `baseNote` and `scale` within the object. And then the function call in audience javascript program will change the scale to C Minor Scale ($C, E\flat, G, C$) starting from middle C, of which midi note number is 60. Whenever a performer send this kind of scale-control messages, they are informed with a message on the top banner that disappear automatically in a few seconds.

By live coding javascript code on the console, the flexibility of what performer can do on the crowd’s machines are expanded indefinitely. For example, a performer can send any kind of text string that can be evaluated in a javascript application in audience side using `script` typed message. See three following examples:

```
publishMessage("audience",type:"script",script:"soundEnabled=false;");
publishMessage("audience",type:"script",script:"refresh();");
publishMessage("audience",type:"script",script:"alert('hello');");
```

The first example will set a variable `soundEnabled` to false, which is a global boolean variable in audience program state to determine whether to switch on/off the sound synthesis (all device will be muted!). The second example will run the function `refresh()`; , which is readily available in the audience program to refresh the page (so everyone is forced to start over!) The third example, maliciously enough, will show an alert box on all smartphones and the web audio synthesis will be halted until the user clicks the okay button.

In orchestrating the crowd, a musician may want to change only the subset of the crowd to produce diverse sound, for example, half the audience playing in C Chord the other half playing F note only. We included `probability` property used with `scale` and `script` typed messages to achieve partial code run. If the performer includes `probability : <a float number>` to the javascript object in messages like above, it will run the parsed action with the probability of the given number. For example, if `probability : 0.5` was attached at the end of object, the code will run with 50% chance so that the performer can make only the (roughly) half of the crowd take the change. The model of one live coder controlling crowd-scale computer networks has been proposed in [68] and we believe this is the first realization of the idea. While the potential of live coding has not yet been explored other than changing the scale, we believe there are novel musical style and aesthetic we can achieve with this live coding large-scale machines. We plan to use this feature to live code the web-based instrument [57] to achieve diverse style of music by changing more than just chord scale (timbre, interface, mapping), while leveraging human computation of individual users for musical expression on this dynamic scenario.

6.5 Validation - Crowd in C in Practice

I presented *Crowd in C* at a series of concerts. The typical audience of the concerts are students, local people in the town, or conference attendees, who have little background either in computer music but are casual smartphone users.

The process of developing the instrument was intermixed with the process of the rehearsing the piece. One common problem in rehearsing audience participation piece is that rehearsals are conducted in a smaller scale than the reality due to the absence of audience. One of the benefit of using the web browser in the rehearsal process was that we could create many instances of participants by having audience interfaces on multiple separate windows

in a powerful computer. We could create several instances of audience per computer without audio stuttering. If it were a native application, the number of instances we can simulate would be limited by the number of devices we own. This is really useful for a musician to experience the collective sound especially because the performer will sit far from audience seats at the actual performance.

For the performance, I composed a short piece and played the role of a performer. The program note included a shortened link (bit.ly and QR code) and a set of step by step instructions that describes how to participate so audience can access the web page and try the interface before the concert begins. Prior to the performance, the performer explained how to participate in the piece with additional slides. As long as participants had devices that can run a web-audio enabled web browser with any connectivity (mobile or WiFi), they were able to participate in the piece. While the cloud service would have used the data center closest to the location of the concert, note that it could have been another data center if the location was in a different country. During the explanation, the audience were guided to launch a web browser (Safari, FireFox or Chrome) on their smart phones and visit the link. There were no additional steps needed such as joining a designated WiFi network, downloading a native app from app stores or typing an IP address, which may be challenging for casual users.

The performance was started by the performer giving sign to audience and pressing "Go-Live" button on the top of the interface. In the beginning, the performer did not intervene to change the global scale for the first few minutes. Rather, he communicated with audience broadcasting chat messages to explain the the instrument (timbre, pitch mapping), to encourage participation, and to introduce what the projection screen showed using the message broadcasting. The performer started to change the scale occasionally for the later part of the performance. The performer interface included frequently used code in a textarea so that the performer can quickly copy and paste in contingency of possible errors in typing code or scale. At certain point, the performer changed the global scale to one note so that all device will generate one pitched sound. This was to play simple melody by quickly running the series of code that had different base note numbers. Later, the crowd was divided into two groups using the `probability` option and one group played a sequence of unified note while the other half played background chord.

6.6 Future Work: Gauging and Understanding Engagement through Performance Analysis

This work has taken practice-based research in which a creative artifact—in this case, a technology-mediated audience participation music piece—is the basis of the contribution to knowledge, but has failed to quantitatively measure its effects in engaging the audience in the performance. It is challenging to validate the extent to which the system effectively engages the audience and facilitates creative collaboration. While subjective methods (e.g., surveys/interviews) give us one way to evaluate system performance, there is the potential for response biases such as acquiescence bias or demand characteristics [235].

To that end, we propose a future work of validating this performance through data-driven methods, and aim to help understand the effectiveness of the system in engaging the audience, as well as the underlying intentions that drive audience participation, whether their motivation is musical or social. Data mining techniques will be applied to the time-evolving networks of audience collaborations to identify effective patterns of interactions, and user preferences that can be adaptively targeted during live performances for increased participation. Our analysis, which will directly rely on the wealth of the data that is generated during live performances, will help create a more concrete (objective), interaction-based notion of what should be supported by future collaborative music systems. In this project, we plan to develop a suite of data-driven computational methods that will help us understand the audience's behaviors during the interactive music performances using analysis of large-scale user-to-user interaction data. The interaction traces of individual audience members can provide important evidences that indicate the extent to which each participant is engaged with the performance. By formulating the interactions between users, their plays, and their musical attributes (e.g., pitch, timbre, duration) as a time-evolving heterogeneous network (as opposed to homogeneous networks that consist of interactions between only the same type of entities), we will devise new network-based techniques to explore how interactions change during a live performance, as audience members explore the system and as the musician orchestrates the audience's efforts in order to shape the piece. The outcome of this work will be an intelligent audience participation system that incorporates findings and methods from live analysis of interaction traces into a novel system that shows the musician the status of audience engagement with real-time analysis, and allows the musician to encourage the audience group who are less active through social interaction during the performance. We propose to perform the piece in a university-wide event as part of the deliverables.

This work will extend our understanding of how participants interact with each other

and generate musical content during live performances. In addition to the methods for scientifically assessing the extent of audience engagement in large-scale participation system, we expect to understand what design components enhance/sustain the level of participatory activities over time and facilitate creativity in content generation. This will lead to insights on how to better facilitate audience engagement in general large-scale participatory system beyond music, e.g., classrooms, public events, and academic conferences. The expected contributions of this work are: Understanding the how social and musical interaction can sustain audience engagement over time Understanding the role of social interaction in facilitating creativity The effects of moderation and intervention in social network evolution Design implications for creating crowd-powered tools for large-scale audience participation in music

6.7 Conclusion

This chapter introduces *Crowd in C*, an audience participation music using a distributed instrument that runs on the web browser using Web Audio API The metaphor of collaborative music making comes from the social interaction model of online dating. The asynchronous nature of participation reduced the difficulty in participating real-time performance as audience members did not need to be always involved in playing. Rather the envisioned interaction resembles the ones in using social media, browsing people's profiles, and interacting with each other. Understanding how collaboration among audience members changed their experience with the performance remains as a intriguing future work. Learning whether the social aspects of the instrument help the audience sustain their interest in participation is important to garner further understanding in prolonged audience engagement.

CHAPTER 7

Codeon: On-Demand Software Development Assistance⁶

So far, I have focused on creating interactive systems that support live collaborative creation. However, a lot of the work that we do is not live. Rather, we collaborate in an asynchronous fashion. This is because as much as we have benefits in real-time collaboration we have benefits in asynchronous collaboration. For example, if two people work separately, they can work in parallel in the time they are available in private environments, yielding better productivity. If it was real-time collaboration, one needs to find time and space that everyone can work together in real time so it may not be best suited for the most efficient style. The question that I wanted to answer is how we can capture and benefit of liveness when we collaborate asynchronously. To that end, I have developed *Codeon*, a programming development environment that supports on-demand assistance from expert crowd workers [63].

7.1 Motivation

In this work, I propose an *asynchronous* on-demand help seeking model for programmers who need support that can be more efficiently provided by remote expert developers than existing methods. I implement and evaluate this model 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 I 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 makes it easy for developers to make sufficiently detailed requests and send to other

⁶Portions of this chapter appear in [63] with the permission of Yan Chen, the lead author of the paper.

developers, making the process quick and effective. Further, Codeon’s asynchronous model is more scalable for multiple helpers than synchronous support tools because it allows multiple helpers to work in parallel with the developer. As the evaluation demonstrates, Codeon supports new forms of parallel collaboration that make remote help-seeking more effective for developers. In this section, I contribute the following:

- an effective approach for integrating external, parallelizable expert assistance into a developer’s on-going process,
- tools and techniques that efficiently preserve liveness of the developers’ requests’ contexts and mediate communication between end users and remote developer helpers,
- evaluations of the trade-offs between speed and accuracy for system components in different help seeking stages,
- a system (Codeon) that instantiates the approach to improve development help-seeking tools, and
- evidence that Codeon helps developers solve more tasks in a given time span than current approaches.

7.2 Related Work

Software developers rely heavily on support from external resources while programming. Although search engines and CQA websites (such as StackOverflow [236]) are the most popular resources for developers, the best support is often provided by other developers [237, 238, 205]. Unlike web-based resources, expert developers can provide personalized help, high-level advice, and project-specific code segments, and can often help identify and overcome bugs that are difficult for a single developer to find on their own [239]. However, it is often prohibitively difficult to find other developers willing to help, particularly for developers working outside of a large organization.

Recently, a small set of paid services began connecting developers with remote expert developers [202, 203], who can provide personalized feedback. These services use a *synchronous*, one-on-one model of communication where developers connect to a remote expert, make a request, and communicate via video chat and a shared editor. However, there are several drawbacks to this synchronous model [205]. There is a coordination cost of finding an expert who is available to help at the right time. If the first expert does not have sufficient expertise (which they cannot know until *after* they connect), there is a

further cost—in both time and money—to finding a new expert. One-on-one mentoring also requires that the developer be attentive to the remote helper throughout the session. Although this is suitable for teaching-oriented requests where a back-and-forth conversation between developers and helpers is desirable, it is less helpful for tasks that can be handed off entirely to the helper, such as requests for short code snippets. This work builds on previous research into pair programming, developer help-seeking, distributed programming, and communication support tools.

7.2.1 Help Seeking in Software Development

7.2.1.1 Community Question Answering

Many Community Question Answering(CQA) websites, such as Stack Overflow [236], provide online asynchronous 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. One of the fundamental problems in CQA systems is a response time to receive an answer after posting the requests. Researchers have taken a various approach to reduce the response time in CQA systems. Prior works [240, 241] have focused on the ephemeral natures of given answers and studied building an *organizational memory* through a growing database of questions and answers. Often times, simple features and limited capabilities of a CQA system could promote distributed help and facilitate forming of social norms in the system, which eventually attract more number of people to participate and accomplish the reduced response time with more answerers [242].

The pipeline of composing a question and getting a response is composed of multiple steps, each of which can contribute to the long waiting time. Previous works often focus on a particular step in the pipeline to expedite the response. For example, expert matching has been a active research topic as finding an 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 [243, 244, 245, 246]. Another approach to reduce the time is to recommend for a set of questions and answers of relevant topics for a given question; this approach may allow a question asker to gain information without waiting for a response from existing questions and answers [247, 248]. In this work, we focus on the very initial step in the pipeline; 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 [249]. 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.

Commercial support platforms, such as Code Mentor [202] and hack.hands() [203], provide more personalized help for software developers. These sites allow developers to create requests and connect them (or let them self-select) with experts, and provide a shared code editor and text/voice communication channel. These sites represent the state of the art for seeking remote help from experts and use synchronous one-on-one communication. In the system evaluation, I show that on-demand support yields similar one-on-one support results while also having the benefit of being parallelizable.

7.2.1.2 Pair Programming

Codeon is related to pair programming [250], 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 [251, 252]. Although the distributed pair programming approach removes many issues in real-time remote collaboration [253], it can still be difficult to coordinate and maintain context in distributed pairs. The system instead aims to automate coordination by temporarily incorporating helpers into a task long enough for them to assist and then move on.

7.2.1.3 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 code base [254]. Ko et al. explored six types of learning barriers in programming systems for beginners and proposed possible solutions from programming system sides [255], and also documented communication among co-located development teams [86]. Guzzi et al. analyzed IDE support for collaboration and evaluated an IDE extension to improve team communication [256].

Whereas these studies of information needs focused on existing team structures, this work 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, I 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.

7.2.1.4 Collaborative Development

Systems like Codeopticon [257] and Codechella [258] 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 [206] 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 [209], and Cloud9 [210] enable users to code collaboratively online in real-time. Although these systems reduce many of the barriers developers face when working at a distance [253] and time spent on environment configuration, they do not support the case when developers are actively seeking help [259]. Codeon allows developers to create requests at any time by speaking their questions while the system automatically captures the problem's context.

7.2.2 IDE-Integrated Help Finding Tools

Codeon is a kind of RSSE [260], which, unlike most CQA websites, often provides relevant information within an IDE. Prior work on RSSE has used knowledge of how developers seek information to develop systems to provide semi-automated support [261, 262]. Blueprint [263] allows developers to rapidly search for a query in an embedded search engine in their local IDE. Seahawk [264] heuristically filtered search results to automatically increase the reliability of search results within an IDE. Hartmann et al. [265] 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. [205] 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 [199] 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 [266, 155, 126]

7.2.3 Human Expert Computation

This work leverages crowdsourcing to make the system available on demand and scalable. By using expert crowd platforms like Upwork [267], which have thousands of developers with a wide range of language and framework expertise, we can hire as many experts as needed to field a developer's set of requests. This allows Codeonto parallelize as much as the end user developer may want to.

Prior work has explored how to use a priori tasking and guidance to automate the coordination and task management process. Foundry [201] 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 this work, I 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.

7.3 Codeon : Implementation

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 a response, and Stage 3) integrating the response (Figure 7.1). 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 editor. It allows developers to make requests (S1) and visualize different formats

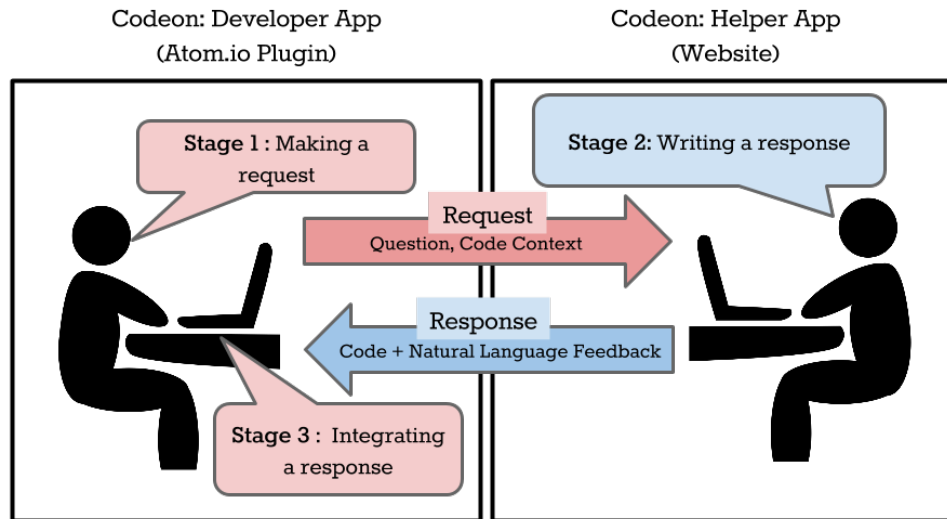


Figure 7.1: Asynchronous interactions between developers and helpers can occur in three stages: making a request (S1), writing a response (S2), and integrating the response (S3). In Codeon, developers use an IDE plug-in to make requests (S1) and integrate responses (S3), and helpers use a web-based IDE to view content and generate responses (S2).

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).

(S1: Making a request) 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. (See Figure. 7.2. 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, we also added a feature that allows developers to add an optional text title for each request for later reference.

(S2: 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 7.3 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 devel-

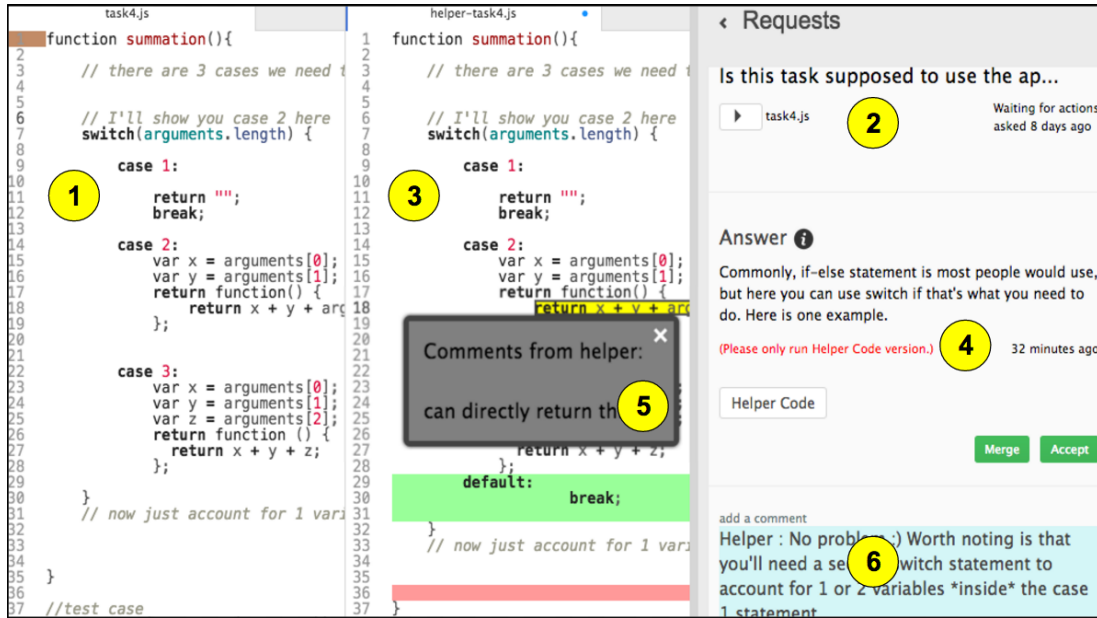


Figure 7.2: Codeon interface where the requests and responses are on the right panel(2). Developer's code(1) and helper's code(3) are side by side for better comparison. Other responses includes explanation(4), annotation(5), and comments(6)

oper'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.

(S3: Response 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. 7.2).

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).

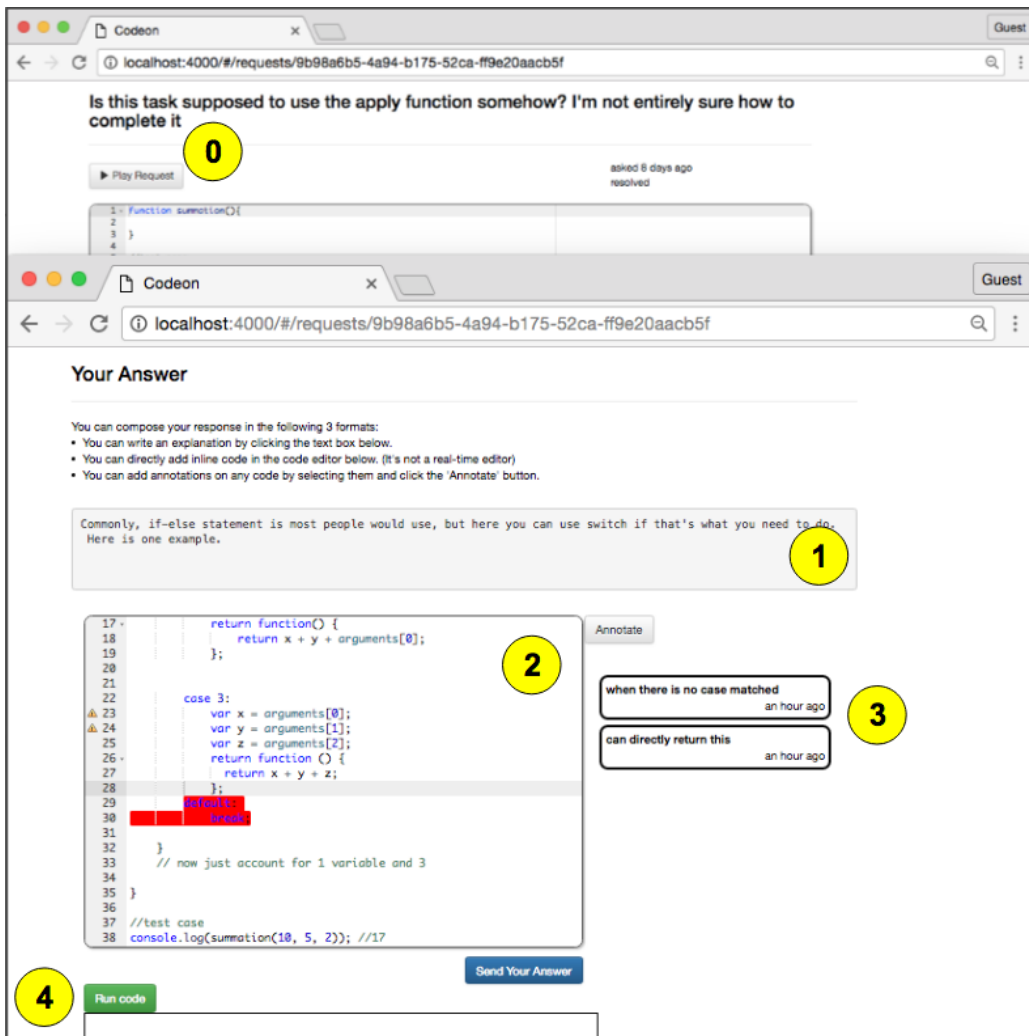


Figure 7.3: The helper side of Codeon is an interactive webpage that allows helpers to replay the request(0) and run the code(4). Helpers can respond to it with explanation(1), and inline code(2), and annotation(3).

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.

7.4 Evaluation

7.4.1 Experimental Setup

Codeon is built for any developers who seek programming support from remote experts. We conducted a laboratory study to better understand how Codeon affects developers’ help-seeking behaviors. 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 [202, 203]. 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



Figure 7.4: Overall result: The # of completed tasks is significantly more in Codeon condition (avg./s.d.).

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).

7.4.2 Results

7.4.2.1 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 7.4 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, $p = 0.03$). To understand *why* developers were more effective with Codeon, we further analyzed our user data, as we will describe in the following sections.

7.4.2.2 Individual Task Performance

To understand the advantages of Codeon, we unpacked our data to investigate participants' performance on each task. As Table 7.1 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 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 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 7.2. 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

	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 7.1: 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 7.2: 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.).

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

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 ($p = 0.05$). 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 additional time coming from social norms, real-time typing process, additional out-of-context questions (or feedback) [268] 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.

7.4.3 Interruptions and Parallelization

Studies have shown that interruptions can be costly to programmers [73]. 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 7.3 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 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 7.4. Table 7.4 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

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 7.4: Time spent and the number of parallelization behavior in two conditions (Avg. / S.D.).

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 shows 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.

7.4.4 Post Interview and Developer Feedback

7.4.4.1 Parallelization

Nearly every developer (11/12) paralleled 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. 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.

7.4.4.2 Interruption

The post-interview reveals 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). 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.

7.4.4.3 Effects of Social Norms

Previous research shows that lower social burden on asynchronous communication activity than synchronous [269]. 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.

“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)”

On the other hand, no one expressed similar concerns for Codeon. I believe Codeon offers a more independent environment with little social pressure and the full control over code and the assistance pipeline.

7.5 Conclusion

In this chapter, we introduced Codeon, an in-IDE tool that allows software developers to get 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. We have already begun using Codeon in our research group to get external help, as well as efficiently collaborate within our own teams. In the future, in-IDE assistance can be used to further improve productivity, reduce interruptions, and even leverage a combination of human and machine intelligence to aid developers.

CHAPTER 8

Live Writing: Preserving Liveness in Asynchronous Written Communication⁷

In this chapter, I present the Live Writing project, which aims to preserve liveness in asynchronous written communication. Live Writing is a web-based application which enable keystroke logging as well as real-time playback of the recorded keystrokes. Writing is an expressive process guided and adapted by thoughts that evolve over time and Live Writing lets a writer record and replay the trajectory revealed in the intermediate stages as it is typed. Here we describe the background in which this idea of live writing is developed, explains motivation and the implementation of the work in progress, proposes new research opportunities exhibited by the notion of live writing.

8.1 Introduction

Writing is a major way for human being to communicate with each other and we live in the age when each individual produces large volumes of written material through digital platforms such as the world wide web, and mobile smart devices. We are in need of presenting writing in communicative and expressive ways and many different forms of written communication have been introduced such as email, social network platforms, blogs, instant messaging, as well as many others. We further motivate the core ideas, and describe the current implementation.

The notion of live writing is directly inspired by the emerging field of live coding [91] in computer music and creative coding. Live coding is audiovisual performance where a programmer writes a program live in front of an audience on stage. The outcome of the program is generative music and perhaps visuals. In live coding music performance, the programming language can be viewed as a musical instrument [92]. It is typical to project

⁷Portions of this chapter appear in [64]

program code text in the performance space for the audience to understand the process of music making. This highly visible nature of the coding process makes the performance aspect of code writing explicit to the audience. This principle is well captured in the following statement of *TOPLAP*¹ (live coding community) manifesto; "Obscurantism is dangerous. Show us your screens." We expand the same idea to writing in general. Writing is an expressive process guided and adapted by thoughts that evolve over time. Showing the entire process of writing as it emerges helps understand the flow of thoughts and provide more expressive power than static final text.

8.2 Motivation

8.2.1 Keystroke Logging and Playback

Keystroke logging has been an important technique in various fields for a long time. Keystroke Level Model (KLM) was an important aspect of research into modeling human performance in many human-computer interaction tasks [270]. More recently it was also used to analyze email organization strategy [271]. Keystroke dynamic has also been used for verifying identities. Habitual temporal patterns of key entries provide a behavioral biometric useful for identifying individuals [272, 273]. Recently a number of important application domains have emerged from studying temporal information from keyboard input. Keystroke logging has been utilized to identify the cognitive state that the user such as the classification of stress conditions [274], recognition of a user's emotional state [275, 276] and detection of affective states when solving math problems [277].

Keystroke logging is a common approach in the field of writing research in order to analyze real-time writing behavior. It provides a non-intrusive and inexpensive technique to monitor user inputs. The writing researchers have developed a number of keystroke logging applications. Inputlog [81] and ScriptLog [82] are such programs used in context of writing research². Most keystroke logging applications include real-time playback recorded keystrokes and it is an effective approach to help subjects account for their writing in retrospect, which is less intrusive than having them think aloud while writing [83]. These applications are mainly for the research purpose of real-time writing analysis rather than for writing in general public.

Expressive playback of text in motion has been realized in kinetic typography [278] with animated effect which is far more powerful in its visual than just rendering text in

¹<http://www.toplap.org/>

²See http://www.writingpro.eu/logging_programs.php for more complete list of keystroke logging programs.

time. Our proposed work differs from kinetic type in that we critically retain the temporal dynamics of typing rather than injecting it in post-production to meet perhaps external such as song-timing information.

8.2.2 Context Recovery and Task Resumption in Collaborative Writing

Collaboration in writing and programming involves challenges that are unique to the collaborative configuration, such as monitoring conflicts and redundancies otherwise, it can cause greater coordinating cost when found later. There exist numerous tools and services that support collaboration on writing and programming together, namely, email exchanges, real-time shared documents, and version control systems. Such technologies remove the physical constraints for people needing to be in the same place at the same time to work together as well as include change awareness so that people can coordinate the work to avoid conflicts, blind modifications, redundancy, and inconsistencies [85]. For example, the `diff` function in version control system or track changes available in modern word processors help collaborators keep track of changes easily. However, these change tracking systems are made of snapshots that users trigger by committing code or exchanging a document over emails. Any intermediate states between transactions are lost thus this approach may require people to have separate in-person discussion later or extra contents to describe the changes such as commit messages and be “on the same page.”

8.2.3 Access to Real-time History of a Document

Version control systems (VCS) such as Git and Subversion are widely used to provide mechanisms for multiple people to work on parts of a shared project and resolve conflicts. The main way that such systems support change awareness is `diff` function, which highlights addition and deletion between user commits. However, as commits are generated at the programmer’s will, information loss between commits can cause change history to be diluted in common task such as refactoring code [90]. Such loss can cause extra costs in teamwork as understanding others progress made (or even their own interrupted tasks) is a significant problem in software engineering team [237]. In addition, preserving temporal dimension in real-time in the replay can provide additional information on the progress. For example, a long pause in activity in writing can be an indicator for task interruption, which has disruptive effects, in task resumption [279]. In the goal of preserving fine-grained history of user interactions have us implement the software integrated in general editing programming

environments, as suggested in [90]. As there are many existing environments, we chose to implement the system as a add-on program that can extend existing software.

8.2.4 Writing as a Real-time Performance Art

The initial purpose of Live Writing was to archive a live coding performance and to expand the idea of live coding to general writing. Typically, live coding performance (or rehearsal) is archived as a screen recording in sync with the music that the program generates. However, the screen recording does not preserve the code text that can be later utilized by other people. This is to support asynchronous collaboration between live coders so that a live coding performance can be reproduced in a symbolic level data. Besides, I explored such artistic, expressive aspects of Live Writing as a real-time audiovisual performance, realizing a textual live-poetry with sonification (See Figure 1.10) [66].

While written communication is indeed an essential way of communicating in various domains, from creative writing to social media, to writing program code. It is a static linear, and asynchronous form of communication since the process of writing occurs before the time at which readers consume the piece of writing. However, the process of writing itself is not static, but dynamic, guided by a writer's thought process that changes over time. The final copy often disguises important information concealed in the process such as: the flow of thoughts, the momentum that accelerated the writing, and spontaneous ideas that were eventually scrapped. The real-time rendering of writing in the same way that it was typed is certainly more expressive than static writing. In addition, a writer can consciously use the record-and-replay function as an expressive dimension. There are various kinds of writer's emotional states (such as contemplation, hesitation, confidence, or agitation) that can emerge during the process of typing based on temporal patterns, for instance, pause, bursts, or corrective steps. The fact that every single entry is shown in the final outcome transforms writing in general into a creative experience in real-time, similar to musical improvisation. For example, a poet can now compose the arrangement of text over time and leverage the articulation of temporal dynamics (such as pauses, bursts, and contemplation or correction) to express his or her own emotional state, revealing the dynamic thought process of a writer to the work's readers. Indeed, such an improvisational nature is prominent on the World Wide Web, for example, vlogging, podcasting, live game streaming, and video tutorials with screen recording and voice over.

8.3 Live Writing - Capturing Liveness of Writing

The idea of preserving liveness of the real-time process in writing is explored in Live Writing system. Live Writing is a software that extends text editors to record a writing session in the finest grains available and enables playback of the session in real time as if the screen was recorded and replayed. The web-based system augments a text editor in a webpage to offer the recording and playback functionalities so that potentially any kind of writing on a web browser can be recorded and reproduced in real time asynchronously. The recording function is hidden in the background and interactions on the editors (keystrokes, clipboard activities, the mouse cursor moves) will be captured with the timestamps. The playback function can be added to the text editors on the web so that it can reproduce the writing session in real-time. This is not a single software or a web application that runs on a specific website but is an API that can potentially extend any text editor available on web browsers to have the recording and playback functions. Live Writing allows users to go back in time and revisit certain states of the copy in the past. An intermediate copy may differ from the final copy and offer writers to retrieve information that was not available in the final document and the previous versions of the document. Live Writing system provides a set of functions that enhances overall user experiences of watching the replay so as to help readers a) watch playback quickly, b) recognize changes spatially and temporally with visualization and animation and c) navigate the temporal dimension quickly to find the information that they want.

Live Writing application is implemented as a web application in order to be easily accessible from the variety of platforms and devices³. The current implementation is crafted with minimalist design; it only has one clean slate; an initial dialog with brief description, screen-sized `<textarea>` object and a few buttons hidden on the side (see 8.1). Once a user press start button, all keystrokes and mouse cursor clicks made inside the `<textarea>` will be recorded in the local machine. And then the user can post a piece of writing when Post button pressed. Once data posted, the user is given a link with which one can access the real-time playback of the writing. The link contains article id and those who writer choose to share the link with can read playback of the writing.

Live Writing is realized in javascript/jQuery/AJAX and node.js. We released the code as an open-source API so that any html `<textarea>` can be extended to feature keystroke logging and playback by including one javascript file. The server runs node.js script which handles static files and store/retrieve recorded keystrokes log in json format. All the other function of keystroke logging and playback is implemented and run in the client machine on

³The application is available at www.echobin.com/.

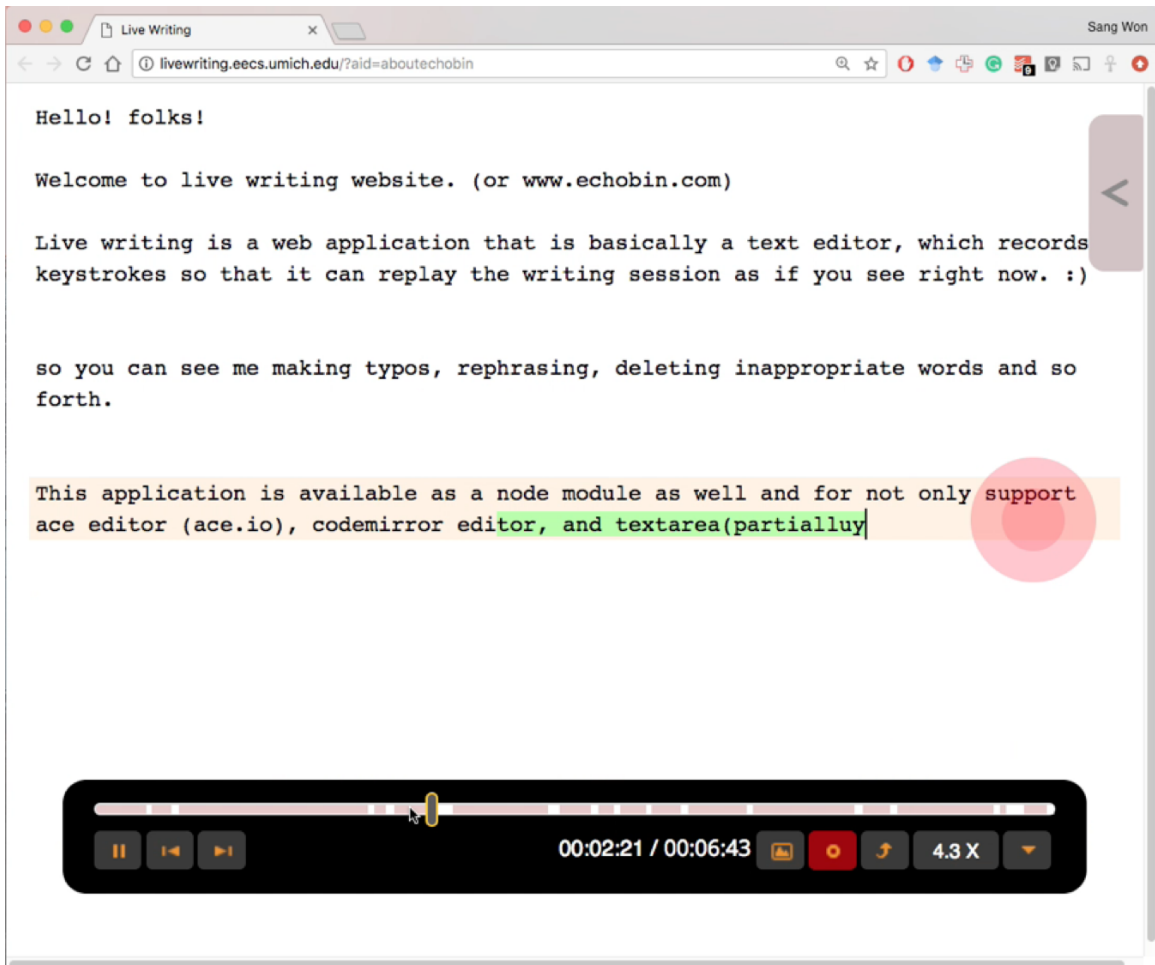


Figure 8.1: Screenshot of Live Writing Website.

which the writer uses. The logged data is not stored in the server until the user chooses to post the writing. Alternatively, one can choose to download the log file instead of posting to the server, if a user wants to keep the logged data locally. Anyone who has the log file will be able to replay the writing by uploading the file. Providing raw data in file will be useful to extract high-level information such as keystroke dynamics or behavioral pattern when processed and we wish that this support researchers to use this as keystroke logging application.

To implement playback, keystroke logged are simulated by specifically reproducing what the keystroke would change in the content of <textarea>. For example, to reproduce a user's pressing 's' key, a web browser does not allow javascript to generate exact same keystroke event for security reason. Instead, it has to append 's' character where the cursor is in the <textarea>at the moment. Note that <textarea>used in keystroke logging is again used for playback. The current implementation only support for English alphabet (EN keyboard) and can be extended to support foreign languages at ease. The improvement on the website is in progress so that a user can customize the writing not only in its visually (font type, size etc.) but also in its playback setting (e.g. playback speed, navigation across time, etc.).

8.4 Future Work

One of the limitations of this work is that it was done in the context of practice based research and lacks the traditional validation process to examine the effects of the system. To that end, it remains as a future work to to I propose a few projects with Live Writing system, which are beyond the scope of the proposed dissertation.

8.4.1 Understanding Effects of Liveness

The goal of this work is to investigate the benefits of presenting the live process to spectators. Live Writing is an appropriate system to examine the idea in the domain of writing and programming. Especially, real-time playback of a writing session can be useful in collaborative setup for context recovery and text comprehension.

In particular, I chose the context of collaborative writing and plan to examine the following hypotheses:

- (H1) Readers would be able to perceive and recall changes better in watching replay than other techniques for change awareness.
- (H2) Watching real-time replay of writing improves reading comprehension.

- (H3) Readers are more engaged when watching Live Writing replay than reading static text.

To measure each hypothesis, I plan to conduct a user study to confirm the benefits of watching a playback of Live Writing in comparison to reading the static text. I plan to recruit online crowdworkers to the following tasks. All the tasks will be implemented as a web application that can be embedded in online crowdsourcing platform.

Task 1: Retention Tasks (H1): Participants will be grouped into three groups (A, B, and C). In this task, participant will read short text. Once they read the text, the actual task will show a revised version of the text presented in three different types of change awareness: A) text with all the changes highlighted — this track changed document is common in commodity word processor software for collaborative writing (e.g. Microsoft Word or Google Docs(See Figure ?? for example), B) real-time replay of someone making the same changes in the text, C) the mix of A) and B), real-time replay of someone making the same changes with the change awareness. Once they understand the revised the text, they will be asked to *undo* to changes made from the revised text to make it close to the original text as much as possible based on how much they recall the changes made in the previous text without access to it. This is to see if watching replay help readers recall the changes made in the context of collaborative writing. Similar experiments to undo changes was conducted in the context of parameter changes in configuration UI to investigate the change awareness of after-glow effects in UI [280]. The task can be repeated more than one time with different text and each text can have different levels of changes - semantic, rephrasing, or grammatical changes. The precision and recall in word level will be calculated for all the submitted text. The completion time of undo task will be also measured. In addition, the time spent on reading the revised text will be also measured to check the efficiency.

Task 2: Tasks for Reading Comprehension and Readers' Engagement(H2 & H3): Participants will be grouped into two groups (A and B). Each group will read the same text in two different ways: A) static text, B) using real-time replay of writing process. Once the finished reading, they will be asked to answer a set of reading comprehension questions. In addition, they will be asked how long they “beleve” it take to read (or watch a replay of) the text (perceived duration). A previous study suggests that increased engagement led to increased perceived speed of time in passing, which in turn increased engagement [281]. Lastly, participants will be asked to evaluate the effectiveness of each text with subjective rating questions to measure readers engagement.

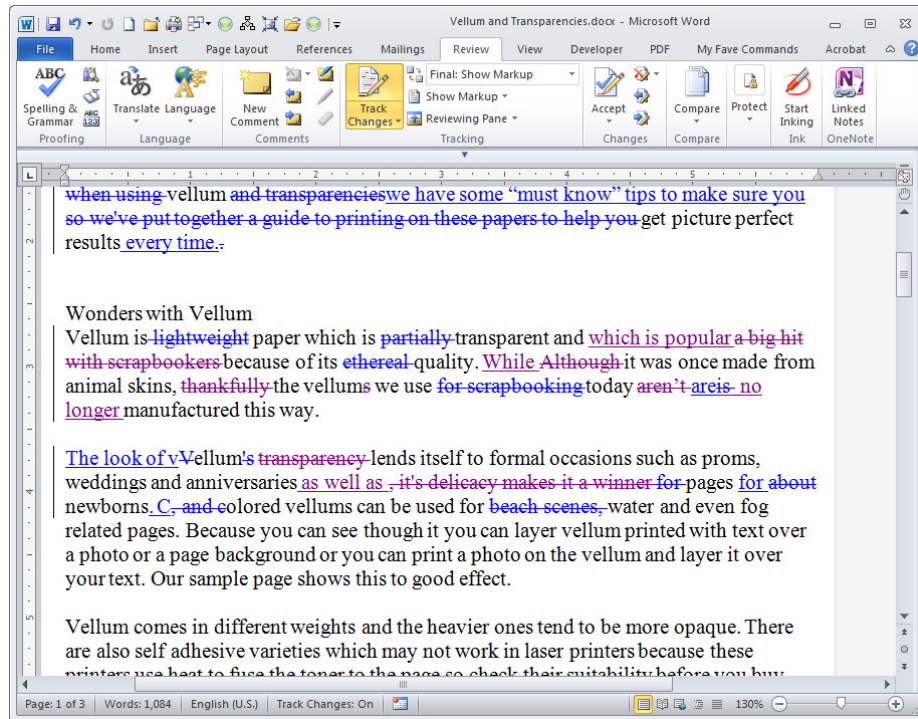


Figure 8.2: Changes are highlighted in Microsoft Words for change awareness in collaborative writing.

8.4.2 Notation Mechanism for Live Coding and Writing

One important future application that we want to explore is the notion of live writing as a notation mechanism in live coding performances. While we described Live Writing in the context of general writing, our original motivation of keystroke logging and playback was to support collaboration in live coding [58]. Our recent survey in networked live coding shows that it is necessary to support asynchronous collaboration and ways to archive/notate live coding music performance are necessary conditions of such collaboration [282]. While one can always record final outcome (sound), videotape the performance and save the program code text produced for the piece, this does not serve as a music notation since the former does not have symbolic information (code), the latter does not include temporal arrangement of code that dynamically changes over time (which code runs when). In this scenario, Live Writing can offer a way to archive a music piece that can serve as a music notation in case of acoustic instruments or MIDI file in case of digital music instruments. We will integrate live writing features in existing live coding environments and evaluate the usage of feature in practice. In addition, we want to make the playback feature not interfere with key entry so that a user can write code text while keystrokes are replayed. This function is to support collaboration with live coders that are not co-located or even with live coders from the past.

In addition, Live Writing system will be deployed in the wild and be enhanced to attract general public to use it regularly. In addition, having ways to embed a Live Writing replay in other web pages will help distribute the replay to be read by more number of readers. Lastly, additional plugins to use the system in commodity code editor (such as Atom.io) can increase the effect of Live Writing especially in the programming context. Similarly, it is desirable to integrate the system into existing live coding systems (such as [99], [283]) to archive live coder's performances and rehearsal. These changes can be followed by further research with the methods of data analysis of large-scale interaction log and target group interviews. The collection of data will be fed to the following project of building intelligent writing systems.

8.4.3 Intelligent Writing Systems

The real-time history of writing can be seen as a signal that changes over time, and this signal can represent the current state of the writing process, similar to eye-tracking data. For example, the length of a document changes in a different way when the writer adds all-new content to it versus when the writer is making minute changes while proofreading. Similarly, code revision patterns can be different when a programmer is debugging code versus when they are refactoring the code.

I propose an intelligent writing environment for collaboration which monitors the live stream of text, understands the individual style of writers and provides meaningful information regarding the status of collaboration to users. The writing system will help a user understand what other writers are working on at high level so that, before the writing diverges, they can initiate communication and resolve problems on the fly. This proposal aims to enhance such group interactivity through two approaches: (1) the analysis of writing in real-time to learn modes and strategies that writers employ and to understand the interdependency of edits made by multiple writers, and (2) exploration and evaluation of live feedback of collaboration that facilitates communication among writers. The Live Writing project provides a source of data with which I can begin to study such patterns for research and later the tool can be extended as an intelligent writing environment.

A piece writing can be seen as a signal that is composed of real-time user interaction (i.e. keystrokes, clipboard events) occurring over time. My hypothesis is that the real-time analysis of writing will reveal important information that reflects the writing strategy of any particular user at the moment and is hidden the final copy. In addition, the hidden changes contained in edits are especially useful in collaborative setups because they can enhance a group's understanding of a writing when the system continually gives users live feedback

and help writers resolve issues before the writing starts to diverge or conflict. I envision an intelligent programming environments that understand writers' real-time activity, recognize meaningful incidents, and provide relevant just-in-time support.

A proper understanding of the temporal dynamics in writing activity has implications in the fields of human computer interaction, computer-supported cooperative work, as well as information retrieval. This method can be used in combination with high-level techniques in other research areas (such as natural language processing, affective computing) to have immediate feedback at the time of interaction. We expand our understanding of the efficacy of having live feedback to a collaborative setup. The liveness in programming environments has been studied only in the context of building immediate connection between a human (programmer) and a machine (the program outcome) Our work on expanding liveness into collaboration promises to be applicable to other collaborative setup based on non-textual interaction.

One of the outcomes of this work is an online writing platform that unlocks for a broader audience the potential of online collaborative writing. We will provide an open-source API for developers to turn any editing environment on the web into a collaborative live-writing environment. In addition, the system will transform a writing into a highly interactive and engaging group experience in various domains: crowd-sourced writing, interactive classroom and real-time performing arts.

CHAPTER 9

Conclusion

In this thesis, I proposed a novel method of involving users in the context of design, creation, and participation: live collaborative creation — directly collaborating with end users in live settings —, exploring the effects of “liveness” in various creative domains. The current body of works in user involvements has challenges in having immediate effects from the methods and engaging the participants involved in the process. Inspired from the participatory process for the purpose of user engagement in creative domains — workshops, class, interactive tutorials, DIY cultures —, I have explored what it takes to computationally support live collaborative creations of end users with creators. The goal of this dissertation was to examine the user involvement method with interactive systems for user involvement by exploring three research questions: R1) identifying and addressing challenges of designing interactive systems that support live collaborative creation, R2) lowering the barriers of it to involve non-expert end users in the setup that requires expertise, and R3) capturing liveness in non-live setups to enhance asynchronous communication and collaboration.

This work documented the interactive systems that I have developed to facilitate live collaboration in creative processes. One of the main contributions is the interactive systems that involve non-expert end users in live settings: 1) a crowd-powered UI design tool that can prototype interactive behaviors [**Chapter 5**] and its challenges involved in real-time communication between creators and crowd workers [**Chapter 3**], 2) a programming environment that visualizes the shared artifact among collaborators (both creators and end-users) and a prototype of a shared programming editors that can help self-coordinate a collaborative task among crowd workers [**Chapter 4**], 3) a performance system for large-scale audience participation in a music concert using mobile ad-hoc network of smartphones [**Chapter 6**], 4) a within-IDE tool for on-demand programming assistance [**Chapter 7**], and 5) a plugin that records and replay written works to capture liveness of writing activity [**Chapter 8**].

Another significant contribution of my dissertation is an exploration of what it means to have “liveness” in designing interactive systems. I provided a definition of liveness that

can be used in the context of collaborative environments. Using this concept, interactive systems can be evaluated in terms of liveness to an extent and can be used to compare to other systems and to enhance liveness in general.

With shared artifacts and communication supported in real time, having liveness in interactive systems help preserve our natural expressivity in using computational systems. For example, SketchExpress utilize the natural expressivity that people have - verbal description and manual demonstration. However, such an expressivity can be easily overlooked when designing a computational system, especially when supporting remote or asynchronous collaboration. In addition, our natural expressivity can be computationally augmented. The expressivity was extended with the computational method of remixing to enable prototyping complex behaviors that cannot be manually demonstrated in person.

Moving forward, I believe liveness can provide an important concept in designing interactive systems for supporting collaboration, enhancing spectator communication, and preserving natural expressivity. In particular, my future research vision includes expanding the notion of liveness and use it towards building *computational systems that facilitate understanding and empathy*. Realizing empathy is suggested not only a way to communicate with others, to design products for users, but also to facilitate creativity [284]. Psychologists typically consider empathy as an individual ability — empathizer’s ability — to share others feelings by observing or learning about their emotional state [285]. Being able to take on the perspective of someone else — a cognitive function—is also part of empathy. However, such perspective can be a compound of all the experiences that a person had over time that may not be accessible to other people. Interactive systems that preserve liveness, which includes continuous visibility, can capture data that is useful to understand other people’s perspective. Live Writing, for example, can capture the continuous trajectory of a writer’s mind that can facilitate understanding in the context of collaborative writing or reading comprehension. On the other hand, in anthropology, empathy is seen as an empathizee’s activity of “an ongoing, dialogical, inter-subjective accomplishment that depends very much on what others are willing or able to let us understand about them” [286]. Therefore, empathizees need a way to effectively express themselves for empathizers to understand their feelings. As aforementioned, I expect that having liveness in computational systems can help preserve and augment the natural expressivity that we have especially in remote and asynchronous setup. With this new understanding of facilitating empathy by having liveness in interactive systems, many problems that we have in social and computational media can be addressed and users can have more engaging experiences, of being connected to others in more profound ways.

All in all, this dissertation represents my journey to bring live, collaborative, and

expressive nature of music-making to broader domains. As music is a real-time temporal art, unfolding collaborative creation over time dimension has been an essential part of the solution to all of the systems that I presented. I believe the notion of live collaboration and live user involvement can be general in broader contexts and wish to expand to other domains; education, social media, productivity tools, DIY culture, and other arts. Lastly, I hope this dissertation will inspire other researchers and practitioners to value liveness, expressivity, and empathy for designing interactive systems.

BIBLIOGRAPHY

- [1] Kujala, S., “User involvement: a review of the benefits and challenges,” *Behaviour & information technology*, Vol. 22, No. 1, 2003, pp. 1–16.
- [2] Kuniavsky, M., *Observing the user experience: a practitioner’s guide to user research*, Morgan kaufmann, 2003.
- [3] Lazar, J., Feng, J. H., and Hochheiser, H., *Research methods in human-computer interaction*, Morgan Kaufmann, 2017.
- [4] Schuler, D. and Namioka, A., *Participatory design: Principles and practices*, CRC Press, 1993.
- [5] Hughes, J. A., Randall, D., and Shapiro, D., “Faltering from ethnography to design,” *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, ACM, 1992, pp. 115–122.
- [6] Dourish, P., “Reading and interpreting ethnography,” *Ways of Knowing in HCI*, Springer, 2014, pp. 1–23.
- [7] Holtzblatt, K. and Jones, S., “Contextual inquiry: A participatory technique for system design,” *Participatory design: Principles and practices*, 1993, pp. 177–210.
- [8] Damodaran, L., “User involvement in the systems design process-a practical guide for users,” *Behaviour & information technology*, Vol. 15, No. 6, 1996, pp. 363–377.
- [9] Juhl, D., “Using field-oriented design techniques to develop consumer software products,” *Field methods casebook for software design*, John Wiley & Sons, Inc., 1996, pp. 215–228.
- [10] Nielsen, J., “Iterative user-interface design,” *Computer*, , No. 11, 1993, pp. 32–41.
- [11] Hughes, J., King, V., Rodden, T., and Andersen, H., “The role of ethnography in interactive systems design,” *interactions*, Vol. 2, No. 2, 1995, pp. 56–65.
- [12] Millen, D. R., “Rapid Ethnography: Time Deepening Strategies for HCI Field Research,” *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS ’00, ACM, New York, NY, USA, 2000, pp. 280–286.

- [13] Holtzblatt, K., Wendell, J. B., and Wood, S., *Rapid contextual design: a how-to guide to key techniques for user-centered design*, Elsevier, 2004.
- [14] Medlock, M. C., Wixon, D., McGee, M., and Welsh, D., “The rapid iterative test and evaluation method: Better products in less time,” *Cost-Justifying Usability (Second edition)*, Elsevier, 2005, pp. 489–517.
- [15] Buur, J. and Bagger, K., “Replacing usability testing with user dialogue,” *Communications of the ACM*, Vol. 42, No. 5, 1999, pp. 63–66.
- [16] Hughes, J., King, V., Rodden, T., and Andersen, H., “Moving out from the control room: ethnography in system design,” *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ACM, 1994, pp. 429–439.
- [17] Schmuckler, M. A., “What is ecological validity? A dimensional analysis,” *Infancy*, Vol. 2, No. 4, 2001, pp. 419–436.
- [18] Sood, S., “Audience involvement and entertainment—Education,” *Communication Theory*, Vol. 12, No. 2, 2002, pp. 153–172.
- [19] Freeman, J., “Extreme sight-reading, mediated expression, and audience participation: Real-time music notation in live performance,” *Computer Music Journal*, Vol. 32, No. 3, 2008, pp. 25–41.
- [20] Bonwell, C. C. and Eison, J. A., *Active Learning: Creating Excitement in the Classroom. 1991 ASHE-ERIC Higher Education Reports.*, ERIC, 1991.
- [21] Hook, J., Schofield, G., Taylor, R., Bartindale, T., McCarthy, J., and Wright, P., “Exploring HCI’s Relationship with Liveness,” *CHI ’12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA ’12, ACM, New York, NY, USA, 2012, pp. 2771–2774.
- [22] Hamilton, W. A., Garretson, O., and Kerne, A., “Streaming on Twitch: Fostering Participatory Communities of Play Within Live Mixed Media,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’14, ACM, New York, NY, USA, 2014, pp. 1315–1324.
- [23] Wang, J., Mughal, M. A., and Juhlin, O., “Experiencing Liveness of a Cherished Place in the Home,” *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, TVX ’15, ACM, New York, NY, USA, 2015, pp. 3–12.
- [24] Hamilton, W. A., Tang, J., Venolia, G., Inkpen, K., Zillner, J., and Huang, D., “Rivulet: Exploring Participation in Live Events through Multi-Stream Experiences,” *Proceedings of the ACM International Conference on Interactive Experiences for TV and Online Video*, ACM, 2016, pp. 31–42.

- [25] Tang, J. C., Venolia, G., and Inkpen, K. M., “Meerkat and Periscope: I Stream, You Stream, Apps Stream for Live Streams,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI ’16, ACM, New York, NY, USA, 2016, pp. 4770–4780.
- [26] Webb, A. M., Wang, C., Kerne, A., and Cesar, P., “Distributed Liveness: Understanding How New Technologies Transform Performance Experiences,” *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ACM, 2016, pp. 432–437.
- [27] Haimson, O. L. and Tang, J. C., “What makes live events engaging on Facebook Live, Periscope, and Snapchat,” *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ACM, 2017, pp. 48–60.
- [28] Willett, N. S., Li, W., Popovic, J., and Finkelstein, A., “Triggering Artwork Swaps for Live Animation,” *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST ’17, ACM, New York, NY, USA, 2017, pp. 85–95.
- [29] Tang, J., Venolia, G., Inkpen, K., Parker, C., Gruen, R., and Pelton, A., “Crowdcasting: Remotely Participating in Live Events Through Multiple Live Streams,” *Proc. ACM Hum.-Comput. Interact.*, Vol. 1, No. CSCW, Dec. 2017, pp. 98:1–98:18.
- [30] Lee, Y.-C., Yen, C.-H., Chiu, P.-T., King, J.-T., and Fu, W.-T., “Tip Me!: Tipping is Changing Social Interactions on Live Streams in China,” *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, ACM, 2018, p. LBW533.
- [31] Lu, Z., Xia, H., Heo, S., and Wigdor, D., “You Watch, You Give, and You Engage: A Study of Live Streaming Practices in China,” *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ACM, 2018, p. 466.
- [32] Siekkinen, M., Favario, L., Masala, E., et al., “Can You See What I See? Quality-of-Experience Measurements of Mobile Live Video Broadcasting,” *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, Vol. 14, No. 2s, 2018, pp. 34.
- [33] Wohn, D. Y., Freeman, G., and McLaughlin, C., “Explaining Viewers’ Emotional, Instrumental, and Financial Support Provision for Live Streamers,” *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ACM, 2018, p. 474.
- [34] Hamilton, W. A., Lupfer, N., Botello, N., Tesch, T., Stacy, A., Merrill, J., Williford, B., Bentley, F. R., and Kerne, A., “Collaborative Live Media Curation: Shared Context for Participation in Online Learning,” *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI ’18, ACM, New York, NY, USA, 2018, pp. 555:1–555:14.

- [35] Conrad, M., ““Fleeting Expletives” and Sports Broadcasts: A Legal Nightmare Needs a Safe Harbor,” *Journal of Legal Aspects of Sport*, Vol. 18, No. 2, 2008, pp. 175–205.
- [36] Ralston, A., Reilly, E. D., and Hemmendinger, D., *Encyclopedia of computer science*, Nature Publishing Group, 2000.
- [37] Brown, S. C. and Knox, D., “Why go to pop concerts? The motivations behind live music attendance,” *Musicae Scientiae*, Vol. 21, No. 3, 2017, pp. 233–249.
- [38] Wikipedia, “Teppanyaki — Wikipedia, The Free Encyclopedia,” <http://en.wikipedia.org/w/index.php?title=Teppanyaki&oldid=810671164>, 2017, [Online; accessed 10-December-2017].
- [39] Auslander, P., *Liveness: Performance in a mediatized culture*, Routledge, 2008.
- [40] Crisell, A., *Liveness and Recording in the Media*, Palgrave Macmillan, 2012.
- [41] Croft, J., “Theses on liveness,” *Organised Sound*, Vol. 12, No. 1, 2007, pp. 59–66.
- [42] Emmerson, S., *Living electronic music*, Routledge, 2017.
- [43] Burdea, G. C., “Force and touch feedback for virtual reality,” 1996.
- [44] Sutcliffe, A., *Multimedia and virtual reality: designing multisensory user interfaces*, Psychology Press, 2003.
- [45] Ranasinghe, N., Nguyen, T. N. T., Liangkun, Y., Lin, L.-Y., Tolley, D., and Do, E. Y.-L., “Vocktail: A Virtual Cocktail for Pairing Digital Taste, Smell, and Color Sensations,” *Proceedings of the 2017 ACM on Multimedia Conference, MM '17*, ACM, New York, NY, USA, 2017, pp. 1139–1147.
- [46] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A., “Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE,” *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '93*, ACM, New York, NY, USA, 1993, pp. 135–142.
- [47] Mendiburu, B., *3D movie making: stereoscopic digital cinema from script to screen*, Focal press, 2012.
- [48] Rumsey, F., *Spatial audio*, Focal Press, 2012.
- [49] Danieau, F., Fleureau, J., Guillotel, P., Mollet, N., Christie, M., and Lécuyer, A., “Toward haptic cinematography: Enhancing movie experience with haptic effects based on cinematographic camera motions,” *IEEE TRANSACTIONS ON Multimedia*, Vol. 21, No. 2, 2014, pp. 11–21.
- [50] Sanders, E. B.-N. and Stappers, P. J., “Co-creation and the new landscapes of design,” *Co-design*, Vol. 4, No. 1, 2008, pp. 5–18.

- [51] Hoyer, W. D., Chandy, R., Dorotic, M., Krafft, M., and Singh, S. S., “Consumer Cocreation in New Product Development,” *Journal of Service Research*, Vol. 13, No. 3, 2010, pp. 283–296.
- [52] Kaulio, M. A., “Customer, consumer and user involvement in product development: A framework and a review of selected methods,” *Total Quality Management*, Vol. 9, No. 1, 1998, pp. 141–149.
- [53] Williamson, O. E., “Transaction-cost economics: the governance of contractual relations,” *The journal of Law and Economics*, Vol. 22, No. 2, 1979, pp. 233–261.
- [54] Lafreniere, B., Grossman, T., Anderson, F., Matejka, J., Kerrick, H., Nagy, D., Vasey, L., Atherton, E., Beirne, N., Coelho, M. H., et al., “Crowdsourced fabrication,” *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, ACM, 2016, pp. 15–28.
- [55] Guo, P. J., Kim, J., and Rubin, R., “How video production affects student engagement: an empirical study of MOOC videos,” *Proceedings of the first ACM conference on Learning@ scale conference*, ACM, 2014, pp. 41–50.
- [56] Lee, S. W., Krosnick, R., Keelean, B., Vaidya, S., O’Keefe, S., Park, S. Y., and Lasecki, W. S., “Exploring Real-time Collaboration in Crowd-powered Systems Through a UI Design Tool.” *Currently in Review*, 2018.
- [57] Lee, S. W. and Essl, G., “Live coding the mobile music instrument,” *Ann Arbor*, Vol. 1001, 2013, pp. 48109–2121.
- [58] Lee, S. W. and Essl, G., “Communication, Control, and State Sharing in Collaborative Live Coding,” *Proceedings of New Interfaces for Musical Expression (NIME)*, London, United Kingdom, 2014.
- [59] Lee, S. W., Chen, Y., Klugman, N., Gouravajhala, S. R., Chen, A., and Lasecki, W. S., “Exploring Coordination Models for Ad Hoc Programming Teams,” *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA ’17, ACM, New York, NY, USA, 2017, pp. 2738–2745.
- [60] Lee, S. W., Zhang, Y., Wong, I., Yiwei., Y., O’Keefe, S., and Lasecki, W. S., “Sketch-Express: Remixing Animations For More Effective Crowd-Powered Prototyping Of Interactive Interfaces,” *Proceedings of the ACM Symposium on User Interface Software and Technology*, UIST, ACM, 2017.
- [61] Lee, S. W. and Freeman, J., “echobo: A mobile music instrument designed for audience to play,” *Ann Arbor*, Vol. 1001, pp. 48109–2121.
- [62] Lee, S. W., de Carvalho Jr, A. D., and Essl, G., “Crowd in C[loud]: Audience participation music with online dating metaphor using cloud service,” 2016.

- [63] Chen, Y., Lee, S. W., Xie, Y., Yang, Y., Lasecki, W. S., and Oney, S., “Codeon: On-Demand Software Development Assistance,” *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI ’17, ACM, New York, NY, USA, 2017, pp. 6220–6231.
- [64] Lee, S. W. and Essl, G., “Live Writing: Asynchronous Playback of Live Coding and Writing,” *Proceedings of International Conference on Live Coding*, Leeds, United Kingdom, 2015.
- [65] Blackwell, A., Cox, G., and Lee, S. W., “Live Writing the Live Coding Book,” *Proceedings of International Conference on Live Coding*, Hamilton, ON, Canada, 2016.
- [66] Lee, S. W., Essl, G., and Martinez, M., “Live Writing : Writing as a Real-time Audiovisual Performance,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Brisbane, Australia, 2016.
- [67] Lee, S. W. and Essl, G., “Web-Based Temporal Typography for Musical Expression and Performance,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, edited by E. Berdahl and J. Allison, Louisiana State University, Baton Rouge, Louisiana, USA, May 31 – June 3 2015, pp. 65–69.
- [68] Lee, S. W. and Essl, G., “Models and Opportunities for Networked Live Coding,” *Ann Alee2016crowdrbor*, Vol. 1001, pp. 48109–2121.
- [69] de Carvalho Junior, A. D., Lee, S. W., and Essl, G., “Understanding Cloud Support for the Audience Participation Concert Performance of Crowd in C[loud],” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Vol. 16 of 2220-4806, Queensland Conservatorium Griffith University, Brisbane, Australia, 2016, pp. 176–181.
- [70] Tanimoto, S. L., “VIVA: A visual language for image processing,” *Journal of Visual Languages & Computing*, Vol. 1, No. 2, 1990, pp. 127–139.
- [71] Tanimoto, S. L., “A perspective on the evolution of live programming,” *Live Programming (LIVE), 2013 1st International Workshop on*, IEEE, 2013, pp. 31–34.
- [72] Oney, S., Myers, B., and Brandt, J., “InterState: a language and environment for expressing interface behavior,” *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, 2014, pp. 263–272.
- [73] Iqbal, S. T. and Horvitz, E., “Disruption and recovery of computing tasks: field study, analysis, and directions,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2007, pp. 677–686.
- [74] Parnin, C. and DeLine, R., “Evaluating cues for resuming interrupted programming tasks,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 93–102.

- [75] Hattori, L., D'Ambros, M., Lanza, M., and Lungu, M., "Software evolution comprehension: Replay to the rescue," *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, IEEE, 2011, pp. 161–170.
- [76] Yoon, Y., Myers, B. A., and Koo, S., "Visualization of fine-grained code change history," *Visual Languages and Human-Centric Computing (VL/HCC), 2013 IEEE Symposium on*, IEEE, 2013, pp. 119–126.
- [77] Simmons, C., "CodeSkimmer: a novel visualization tool for capturing, replaying, and understanding fine-grained change in software," 2013.
- [78] Viégas, F. B., Wattenberg, M., and Dave, K., "Studying cooperation and conflict between authors with history flow visualizations," *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2004, pp. 575–582.
- [79] Suh, B., Chi, E. H., Kittur, A., and Pendleton, B. A., "Lifting the veil: improving accountability and social transparency in Wikipedia with wikidashboard," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2008, pp. 1037–1040.
- [80] Chevalier, F., Dragicevic, P., Bezerianos, A., and Fekete, J.-D., "Using text animated transitions to support navigation in document histories," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 683–692.
- [81] Leijten, M. and Van Waes, L., "Inputlog: New perspectives on the logging of on-line writing processes in a Windows environment," *Studies in writing*, Vol. 18, 2006, pp. 73.
- [82] Strömquist, S., Holmqvist, K., Johansson, V., Karlsson, H., and Wengelin, Å., "What keystroke-logging can reveal about writing," *Computer key-stroke logging and writing: methods and applications (Studies in Writing)*, Vol. 18, 2006, pp. 45–72.
- [83] Latif, M. M. A., "A state-of-the-art review of the real-time computer-aided study of the writing process," *IJES, International Journal of English Studies*, Vol. 8, No. 1, 2008, pp. 29–50.
- [84] Dourish, P. and Bellotti, V., "Awareness and coordination in shared workspaces," *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, ACM, 1992, pp. 107–114.
- [85] Tam, J. and Greenberg, S., "A framework for asynchronous change awareness in collaborative documents and workspaces," *International Journal of Human-Computer Studies*, Vol. 64, No. 7, 2006, pp. 583–598.
- [86] Ko, A. J., DeLine, R., and Venolia, G., "Information needs in collocated software development teams," *Proceedings of the 29th international conference on Software Engineering*, IEEE Computer Society, 2007, pp. 344–353.

- [87] Tam, J., McCaffrey, L., Maurer, F., and Greenberg, S., “Change awareness in software engineering using two dimensional graphical design and development tools,” 2000.
- [88] Noël, S. and Robert, J.-M., “Empirical study on collaborative writing: What do co-authors do, use, and like?” *Computer Supported Cooperative Work (CSCW)*, Vol. 13, No. 1, 2004, pp. 63–89.
- [89] Bach, B., Pietriga, E., and Fekete, J.-D., “GraphDiaries: animated transitions and temporal navigation for dynamic networks,” *Visualization and Computer Graphics, IEEE Transactions on*, Vol. 20, No. 5, 2014, pp. 740–754.
- [90] Robbes, R. and Lanza, M., “A change-based approach to software evolution,” *Electronic Notes in Theoretical Computer Science*, Vol. 166, 2007, pp. 93–109.
- [91] Collins, N., McLean, A., Rohrhuber, J., and Ward, A., “Live coding in laptop performance,” *Organised Sound*, Vol. 8, No. 03, 2003, pp. 321–330.
- [92] Blackwell, A. and Collins, N., “The programming language as a musical instrument,” *Proceedings of PPIG05 (Psychology of Programming Interest Group)*, 2005.
- [93] Brown, A. R. and Sorensen, A. C., “aa-cell in practice: An approach to musical live coding,” *Proceedings of the International Computer Music Conference*, International Computer Music Association, 2007, pp. 292–299.
- [94] Freeman, J. and Troyer, A., “Collaborative textual improvisation in a laptop ensemble,” *Computer Music Journal*, Vol. 35, No. 2, 2011, pp. 8–21.
- [95] Ogborn, D., “Live Coding in a Scalable, Participatory Laptop Orchestra,” *Computer Music Journal*, Vol. 38, No. 1, 2014, pp. 17–30.
- [96] Sorensen, A. C., “A distributed memory for networked livecoding performance,” *Proceedings of the International Computer Music Conference*, 2010, pp. 530–533.
- [97] Wilson, S., Lorway, N., Coull, R., Vasilakos, K., and Moyers, T., “Free as in BEER: Some Explorations into Structured Improvisation Using Networked Live-Coding Systems,” *Computer Music Journal*, Vol. 38, No. 1, 2014, pp. 54–64.
- [98] Dannenberg, R. B., Cavaco, S., Ang, E., Avramovic, I., Aygun, B., Baek, J., Barn-dollar, E., Duterte, D., Grafton, J., Hunter, R., et al., “The Carnegie Mellon Laptop Orchestra,” 2007.
- [99] Roberts, C. and Kuchera-Morin, J., “Gibber: Live Coding Audio In The Browser,” *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, 2012.
- [100] “Sketchpad,” <http://sketchpad.cc/>, Accessed: 2014-07.
- [101] Lee, S. W. and Freeman, J., “Real-Time Music Notation in Mixed Laptop Acoustic Ensembles,” *Computer Music Journal*, Vol. 37, No. 4, Dec 2013, pp. 24–36.

- [102] McLean, A., Griffiths, D., Collins, N., and Wiggins, G., “Visualisation of live code,” *Proceedings of Electronic Visualisation and the Arts 2010*, 2010.
- [103] Rohruber, J. and Campo, A. d., “The republic quark,” <https://github.com/supercollider-quarks/Republic>, 2011.
- [104] McKinney, C., “Quick Live Coding Collaboration In The Web Browser,” *Proceedings of New Interfaces for Musical Expression*, London, U.K., 2014.
- [105] Swift, B., Sorensen, A. C., Gardner, H., and Hosking, J., “Visual code annotations for cyberphysical programming,” *1st International Workshop on Live Programming (LIVE)*, IEEE, 2013.
- [106] Brown, C. and Bischoff, J., “Indigenous to the Net: early network music bands in the San Francisco Bay area,” *Available at crossfade. walkerart.org/brownbischoff*, 2002.
- [107] Ogborn, D., “EspGrid: A Protocol for Participatory Electronic Ensemble Performance,” *Audio Engineering Society Convention 133*, Audio Engineering Society, 2012.
- [108] “Amazon’s Mechanical Turk,” 2017, <http://www.mturk.com>.
- [109] Lease, M., Hullman, J., Bigham, J. P., Bernstein, M. S., Kim, J., Lasecki, W. S., Bakhshi, S., Mitra, T., and Miller, R. C., “Mechanical turk is not anonymous,” 2013.
- [110] Little, G., Chilton, L. B., Goldman, M., and Miller, R. C., “TurKit: human computation algorithms on mechanical turk,” *User Interface Software and Technology*, UIST, 2010, pp. 57–66.
- [111] Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D., and Panovich, K., “Soylent: a word processor with a crowd inside,” *User Interface Software and Technology*, UIST, 2010, pp. 313–322.
- [112] Nebeling, M., To, A., Guo, A., de Freitas, A. A., Teevan, J., Dow, S. P., and Bigham, J. P., “WearWrite: Crowd-Assisted Writing from Smartwatches,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI ’16, ACM, New York, NY, USA, 2016, pp. 3834–3846.
- [113] Teevan, J., Iqbal, S. T., and von Veh, C., “Supporting Collaborative Writing with Microtasks,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI ’16, ACM, New York, NY, USA, 2016, pp. 2657–2668.
- [114] Kim, J., Cheng, J., and Bernstein, M. S., “Ensemble: Exploring Complementary Strengths of Leaders and Crowds in Creative Collaboration,” *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW ’14, ACM, New York, NY, USA, 2014, pp. 745–755.

- [115] Kittur, A., Suh, B., Pendleton, B. A., and Chi, E. H., “He Says, She Says: Conflict and Coordination in Wikipedia,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, ACM, New York, NY, USA, 2007, pp. 453–462.
- [116] Kittur, A., Smus, B., and Kraut, R., “CrowdForge: Crowdsourcing Complex Work,” Tech. Rep. CMUHCII-11-100, Carnegie Mellon University, 2011.
- [117] Lasecki, W. S., Murray, K. I., White, S., Miller, R. C., and Bigham, J. P., “Real-time Crowd Control of Existing Interfaces,” *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM, New York, NY, USA, 2011, pp. 23–32.
- [118] Lasecki, W. S., Thiha, P., Zhong, Y., Brady, E., and Bigham, J. P., “Answering Visual Questions with Conversational Crowd Assistants,” *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '13, ACM, New York, NY, USA, 2013, pp. 18:1–18:8.
- [119] Bigham, J. P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R. C., Miller, R., Tatarowicz, A., White, B., White, S., and Yeh, T., “VizWiz: nearly real-time answers to visual questions,” *User Interface Software and Technology*, UIST, 2010, pp. 333–342.
- [120] Bernstein, M. S., Brandt, J., Miller, R. C., and Karger, D. R., “Crowds in two seconds: Enabling realtime crowd-powered interfaces,” *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011, pp. 33–42.
- [121] Gordon, M., Bigham, J. P., and Lasecki, W. S., “LegionTools: A Toolkit + UI for Recruiting and Routing Crowds to Synchronous Real-Time Tasks,” *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15 Adjunct, ACM, New York, NY, USA, 2015, pp. 81–82.
- [122] Gouravajhala, S. R., Yim, J., Desingh, K., Huang, Y., Jenkins, O. C., and Lasecki, W. S., “EURECA: Enhanced Understanding of Real Environments via Crowd Assistance,” 2018.
- [123] Bernstein, M. S., Brandt, J., Miller, R. C., and Karger, D. R., “Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces,” *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM, New York, NY, USA, 2011, pp. 33–42.
- [124] Lasecki, W. S., Murray, K. I., White, S., Miller, R. C., and Bigham, J. P., “Real-time Crowd Control of Existing Interfaces,” *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, ACM, New York, NY, USA, 2011, pp. 23–32.
- [125] Huang, T. K., Lasecki, W. S., Azaria, A., and Bigham, J. P., ““Is There Anything Else I Can Help You With?” Challenges in Deploying an On-Demand Crowd-Powered

Conversational Agent,” *Fourth AAAI Conference on Human Computation and Crowdsourcing*, 2016.

- [126] Lasecki, W. S., Wesley, R., Nichols, J., Kulkarni, A., Allen, J. F., and Bigham, J. P., “Chorus: A Crowd-powered Conversational Assistant,” *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST ’13, ACM, New York, NY, USA, 2013, pp. 151–162.
- [127] Andolina, S., Schneider, H., Chan, J., Klouche, K., Jacucci, G., and Dow, S., “Crowd-board: Augmenting In-Person Idea Generation with Real-Time Crowds,” *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition*, C&C ’17, ACM, New York, NY, USA, 2017, pp. 106–118.
- [128] Hosio, S., Goncalves, J., van Berkel, N., Klakegg, S., Konomi, S., and Kostakos, V., “Facilitating Collocated Crowdsourcing on Situated Displays,” *Human-Computer Interaction*, 2017, pp. 1–37.
- [129] Dow, S., Kulkarni, A., Klemmer, S., and Hartmann, B., “Shepherding the Crowd Yields Better Work,” *Computer Supported Cooperative Work*, CSCW, 2012, pp. 1013–1022.
- [130] Luther, K., Pavel, A., Wu, W., Tolentino, J.-I., Agrawala, M., Hartmann, B., and Dow, S. P., “CrowdCrit: crowdsourcing and aggregating visual design critique,” *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*, ACM, 2014, pp. 21–24.
- [131] Yuan, A., Luther, K., Krause, M., Vennix, S. I., Dow, S. P., and Hartmann, B., “Almost an expert: The effects of rubrics and expertise on perceived value of crowdsourced design critiques,” *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, ACM, 2016, pp. 1005–1017.
- [132] Retelny, D., Robaszkiewicz, S., To, A., Lasecki, W. S., Patel, J., Rahmati, N., Doshi, T., Valentine, M., and Bernstein, M. S., “Expert Crowdsourcing with Flash Teams,” *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST ’14, ACM, New York, NY, USA, 2014, pp. 75–85.
- [133] Valentine, M. A., Retelny, D., To, A., Rahmati, N., Doshi, T., and Bernstein, M. S., “Flash Organizations: Crowdsourcing Complex Work by Structuring Crowds As Organizations,” *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI ’17, ACM, New York, NY, USA, 2017, pp. 3523–3537.
- [134] Salehi, N., McCabe, A., Valentine, M., and Bernstein, M., “Huddler: Convening Stable and Familiar Crowd Teams Despite Unpredictable Availability,” *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW ’17, ACM, New York, NY, USA, 2017, pp. 1700–1713.
- [135] Salehi, N., Teevan, J., Iqbal, S., and Kamar, E., “Communicating Context to the Crowd for Complex Writing Tasks,” *Proceedings of the 2017 ACM Conference on*

Computer Supported Cooperative Work and Social Computing, CSCW '17, ACM, New York, NY, USA, 2017, pp. 1890–1901.

- [136] Hasse, J., “Moths,” *Visible Music, Euclid, OH*, 1986.
- [137] Taylor, B., “A History of the Audience as a Speaker Array,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Aalborg University Copenhagen, Copenhagen, Denmark, 2017, pp. 481–486.
- [138] Levin, G. et al., “Dialtones-a telesymphony,” 2001.
- [139] McAllister, G., Alcorn, M., and Strain, P., “Interactive performance with wireless PDAs,” *Proceedings of the International Computer Music Conference (ICMC)*, 2004.
- [140] Tanaka, A. and Gemeinboeck, P., “Net_Dérive: Conceiving and producing a locative media artwork,” *Mobile Technologies: From Telecommunications to Media*. Routledge, 2008.
- [141] Oh, J. and Wang, G., “Audience-participation techniques based on social mobile computing,” *Proceedings of the International Computer Music Conference (ICMC)*, 2011.
- [142] Dahl, L., Herrera, J., and Wilkerson, C., “TweetDreams: Making Music with the Audience and the World using Real-time Twitter Data.” *NIME*, 2011, pp. 272–275.
- [143] Rogers, C., “Web audio API,” 2012.
- [144] Monschke, J., “Building a Collaborative Digital Audio Workstation Based on the Web Audio API,” *WAC - 1st Web Audio Conference*, 2015.
- [145] Allison, J., “Traversal,” *WAC - 1st Web Audio Conference*, 2015.
- [146] Kita, T., “Smartphone Jam Session with Audience,” *WAC - 1st Web Audio Conference*, 2015.
- [147] Robaszekiewicz, S. and Schnell, N., “Soundworks—A playground for artists and developers to create collaborative mobile web performances,” 2015.
- [148] Piquemal, S. and Shaw, T., “Field #2,” *WAC - 1st Web Audio Conference*, 2015.
- [149] Taylor, B., “Pearl River (2013 2015),” *WAC - 1st Web Audio Conference*, 2015.
- [150] Tang, J. C., “Findings from observational studies of collaborative work,” *International Journal of Man-machine studies*, Vol. 34, No. 2, 1991, pp. 143–160.
- [151] Gutwin, C. and Greenberg, S., “A descriptive framework of workspace awareness for real-time groupware,” *Computer Supported Cooperative Work (CSCW)*, Vol. 11, No. 3-4, 2002, pp. 411–446.

- [152] Fussell, S. R., Kraut, R. E., and Siegel, J., “Coordination of Communication: Effects of Shared Visual Context on Collaborative Work,” *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work, CSCW '00*, ACM, New York, NY, USA, 2000, pp. 21–30.
- [153] Lasecki, W. S. and Bigham, J. P., “Interactive crowds: Real-time crowdsourcing and crowd agents,” *Handbook of human computation*, Springer, 2013, pp. 509–521.
- [154] Kittur, A., Nickerson, J. V., Bernstein, M., Gerber, E., Shaw, A., Zimmerman, J., Lease, M., and Horton, J., “The Future of Crowd Work,” *Proceedings of the 2013 Conference on Computer Supported Cooperative Work, CSCW '13*, ACM, New York, NY, USA, 2013, pp. 1301–1318.
- [155] Lasecki, W. S., Kim, J., Rafter, N., Sen, O., Bigham, J. P., and Bernstein, M. S., “Apparition: Crowdsourced User Interfaces That Come to Life As You Sketch Them,” *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI '15*, ACM, New York, NY, USA, 2015, pp. 1925–1934.
- [156] Gutwin, C. and Greenberg, S., “A descriptive framework of workspace awareness for real-time groupware,” *Computer Supported Cooperative Work (CSCW)*, Vol. 11, No. 3-4, 2002, pp. 411–446.
- [157] Bergkvist, A., Burnett, D. C., Jennings, C., and Narayanan, A., “Webtrc 1.0: Real-time communication between browsers,” *Working draft, W3C*, Vol. 91, 2012.
- [158] Tang, J. C., “Listing, drawing and gesturing in design: A study of the use of shared workspaces by design teams,” *Ph.D. Thesis*, 1989.
- [159] Karsenty, L., “Cooperative work and shared visual context: An empirical study of comprehension problems in side-by-side and remote help dialogues,” *Human-Computer Interaction*, Vol. 14, No. 3, 1999, pp. 283–315.
- [160] Kraut, R. E., Fussell, S. R., and Siegel, J., “Visual Information As a Conversational Resource in Collaborative Physical Tasks,” *Hum.-Comput. Interact.*, Vol. 18, No. 1, June 2003, pp. 13–49.
- [161] Reid, F. J. and Reed, S. E., “Conversational grounding and visual access in collaborative design,” *CoDesign*, Vol. 3, No. 2, 2007, pp. 111–122.
- [162] Cicchetti, D. V., “Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology,” *Psychological Assessment*, 6(4), 1994, pp. 284–290.
- [163] Nielsen, J. and Landauer, T. K., “A mathematical model of the finding of usability problems,” *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, ACM, 1993, pp. 206–213.

- [164] Tang, A., Pahud, M., Inkpen, K., Benko, H., Tang, J. C., and Buxton, B., “Three’s company: understanding communication channels in three-way distributed collaboration,” *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, ACM, 2010, pp. 271–280.
- [165] Scott, S. D., Carpendale, M. S. T., and Inkpen, K. M., “Territoriality in collaborative tabletop workspaces,” *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, ACM, 2004, pp. 294–303.
- [166] “Voice Qualities,” 2017, <http://www.ncvs.org/ncvs/tutorials/voiceprod/tutorial/quality.html>.
- [167] Hinds, P. J., “The curse of expertise: The effects of expertise and debiasing methods on prediction of novice performance.” *Journal of Experimental Psychology: Applied*, Vol. 5, No. 2, 1999, pp. 205.
- [168] Gergle, D., Kraut, R. E., and Fussell, S. R., “Action As Language in a Shared Visual Space,” *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*, CSCW ’04, ACM, New York, NY, USA, 2004, pp. 487–496.
- [169] Isaacs, E. A. and Clark, H. H., “References in conversation between experts and novices.” *Journal of experimental psychology: general*, Vol. 116, No. 1, 1987, pp. 26.
- [170] Oviatt, S., Cohen, P., Wu, L., Duncan, L., Suhm, B., Bers, J., Holzman, T., Winograd, T., Landay, J., Larson, J., et al., “Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions,” *Human-computer interaction*, Vol. 15, No. 4, 2000, pp. 263–322.
- [171] Gutwin, C., “The effects of network delays on group work in real-time groupware,” *ECSCW 2001*, Springer, 2001, pp. 299–318.
- [172] Recktenwald, D., “Toward a transcription and analysis of live streaming on Twitch,” *Journal of Pragmatics*, Vol. 115, 2017, pp. 68 – 81.
- [173] Clark, H. H., Brennan, S. E., et al., “Grounding in communication,” *Perspectives on socially shared cognition*, Vol. 13, No. 1991, 1991, pp. 127–149.
- [174] Greenberg, S. and Marwood, D., “Real time groupware as a distributed system: concurrency control and its effect on the interface,” *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ACM, 1994, pp. 207–217.
- [175] Lee, S. W., Chen, Y., and Lasecki, W. S., “The Need for Real-Time Crowd Generation of Task Lists from Speech,” *Work-in-Progress in The fifth AAI Conference on Human Computation and Crowdsourcing*, HCOMP, 2017.
- [176] Janarthanam, S. and Lemon, O., “Learning to adapt to unknown users: referring expression generation in spoken dialogue systems,” *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2010, pp. 69–78.

- [177] Morris, M. R., Ryall, K., Shen, C., Forlines, C., and Vernier, F., “Beyond social protocols: Multi-user coordination policies for co-located groupware,” *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, ACM, 2004, pp. 262–265.
- [178] Ellis, C. A., Gibbs, S. J., and Rein, G., “Groupware: some issues and experiences,” *Communications of the ACM*, Vol. 34, No. 1, 1991, pp. 39–58.
- [179] Gutwin, C. and Penner, R., “Improving Interpretation of Remote Gestures with Telepointer Traces,” *Proceedings of the 2002 ACM Conference on Computer Supported Cooperative Work*, CSCW '02, ACM, New York, NY, USA, 2002, pp. 49–57.
- [180] Lasecki, W. S., Gordon, M., Leung, W., Lim, E., Bigham, J. P., and Dow, S. P., “Exploring privacy and accuracy trade-offs in crowdsourced behavioral video coding,” *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 2015, pp. 1945–1954.
- [181] Kaur, H., Gordon, M., Yang, Y., Bigham, J. P., Teevan, J., Kamar, E., and Lasecki, W. S., “Crowdmask: Using crowds to preserve privacy in crowd-powered systems via progressive filtering,” *Proceedings of the AAAI Conference on Human Computation (HCOMP 2017)*, HCOMP, Vol. 17, 2017.
- [182] Lee, S. W. and Essl, G., “Communication, Control, and State Sharing in Collaborative Live Coding,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Goldsmiths, University of London, London, United Kingdom, 2014, pp. 263–268.
- [183] Essl, G., “UrMus – An Environment for Mobile Instrument Design and Performance,” *Proceedings of the International Computer Music Conference (ICMC)*, Stony Brooks/New York, June 1-5 2010.
- [184] Cook, P., “Principles for designing computer music controllers,” *Proceedings of the 2001 conference on New interfaces for musical expression*, National University of Singapore, 2001, pp. 1–4.
- [185] Murray-Browne, T., Mainstone, D., Bryan-Kinns, N., and Plumbley, M. D., “The medium is the message: Composing instruments and performing mappings,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2011, pp. 56–59.
- [186] Magnusson, T., “Designing constraints: Composing and performing with digital musical systems,” *Computer Music Journal*, Vol. 34, No. 4, 2010, pp. 62–73.
- [187] Roberts, C. and Hollerer, T., “Composition for conductor and audience: new uses for mobile devices in the concert hall,” *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*, ACM, 2011, pp. 65–66.

- [188] Weitzner, N., Freeman, J., Chen, Y.-L., and Garrett, S., “massMobile: towards a flexible framework for large-scale participatory collaborations in live performances,” *Organised Sound*, Vol. 18, No. 01, 2013, pp. 30–42.
- [189] Linson, A., “Unnecessary constraints: a challenge to some assumptions of digital musical instrument design,” *Proceedings of the International Computer Music Conference (ICMC)*, 2011.
- [190] Wang, G., Misra, A., Davidson, P., and Cook, P. R., “CoAudicle: A Collaborative Audio Programming Space,” *In Proceedings of the International Computer Music Conference*, Citeseer, 2005.
- [191] Weinberg, G., “Interconnected musical networks: Toward a theoretical framework,” *Computer Music Journal*, Vol. 29, No. 2, 2005, pp. 23–39.
- [192] Ierusalimsky, R., *Programming in lua*, Roberto Ierusalimsky, 2006.
- [193] Brooks, F. P., *The mythical man-month*, Vol. 1995, Addison-Wesley Reading, MA, 1975.
- [194] Surowiecki, J., *The wisdom of crowds*, Anchor, 2005.
- [195] Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E., “Crowdforge: Crowdsourcing complex work,” *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011, pp. 43–52.
- [196] Lasecki, W. S., Homan, C., and Bigham, J. P., “Architecting real-time crowd-powered systems,” *Human Computation*, Vol. 1, No. 1, 2014.
- [197] LaToza, T. D. and van der Hoek, A., “Crowdsourcing in Software Engineering: Models, Motivations, and Challenges,” *Software, IEEE*, Vol. 33, No. 1, 2016, pp. 74–80.
- [198] Lakhani, K. R., Garvin, D. A., and Lonstein, E., “Topcoder (a): Developing software through crowdsourcing,” 2010.
- [199] LaToza, T. D., Towne, W. B., Adriano, C. M., and van der Hoek, A., “Microtask programming: Building software with a crowd,” *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, 2014, pp. 43–54.
- [200] Stol, K.-J. and Fitzgerald, B., “Two’s company, three’s a crowd: a case study of crowdsourcing software development,” *Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 187–198.
- [201] Retelny, D., Robaszkiewicz, S., To, A., Lasecki, W. S., Patel, J., Rahmati, N., Doshi, T., Valentine, M., and Bernstein, M. S., “Expert crowdsourcing with flash teams,” *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, 2014, pp. 75–85.

- [202] Codementor Inc., “Code Mentor, <https://codementor.io/>,” 2014, Accessed: April, 2016.
- [203] HackHands Inc, “Hack.hands(), <https://hackhands.com/>,” 2015, Accessed: April, 2016.
- [204] Crowdbotics., 2014, Accessed: January, 2017.
- [205] Chen, Y., Oney, S., and Lasecki, W. S., “Towards Providing On-Demand Expert Support for Software Developers,” *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ACM, 2016, pp. 3192–3203.
- [206] Goldman, M., Little, G., and Miller, R. C., “Real-time collaborative coding in a web IDE,” *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM, 2011, pp. 155–164.
- [207] Ho, C., Raha, S., Gehringer, E., and Williams, L., “Sangam: a distributed pair programming plug-in for Eclipse,” *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, ACM, 2004, pp. 73–77.
- [208] Salinger, S., Oezbek, C., Beecher, K., and Schenk, J., “Saros: an eclipse plug-in for distributed party programming,” *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, ACM, 2010, pp. 48–55.
- [209] Sinan Yasar, D. Y., “Koding,” 2012, Accessed: April, 2016.
- [210] Cloud9 IDE Inc., “Cloud9 IDE, <https://c9.io/>,” 2010, Accessed: April, 2016.
- [211] Glass, R. L., *Facts and fallacies of software engineering*, Addison-Wesley Professional, 2002.
- [212] Landay, J. A. and Myers, B. A., “Interactive Sketching for the Early Stages of User Interface Design,” *Human Factors in Computing Systems*, CHI, ACM Press/Addison-Wesley Publishing Co., 1995, pp. 43–50.
- [213] Lin, J., Newman, M. W., Hong, J. I., and Landay, J. A., “DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design,” *Human Factors in Computing Systems*, CHI, 2000, pp. 510–517.
- [214] Myers, B., Park, S. Y., Nakano, Y., Mueller, G., and Ko, A., “How designers design and program interactive behaviors,” *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sept 2008, pp. 177–184.
- [215] Davis, R. C., Colwell, B., and Landay, J. A., “K-sketch: A ‘Kinetic’ Sketch Pad for Novice Animators,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI ’08, ACM, New York, NY, USA, 2008, pp. 413–422.

- [216] Barnes, C., Jacobs, D. E., Sanders, J., Goldman, D. B., Rusinkiewicz, S., Finkelstein, A., and Agrawala, M., “Video Puppetry: A Performative Interface for Cutout Animation,” *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia ’08, ACM, New York, NY, USA, 2008, pp. 124:1–124:9.
- [217] Sohn, E. and Choy, Y. C., “Sketch-n-Stretch: Sketching Animations Using Cutouts,” *IEEE Computer Graphics and Applications*, Vol. 32, No. 3, May 2012, pp. 59–69.
- [218] Obrenovic, v. and Martens, J.-B., “Sketching Interactive Systems with Sketchify,” *ACM Trans. Comput.-Hum. Interact.*, Vol. 18, No. 1, May 2011, pp. 4:1–4:38.
- [219] Kazi, R. H., Chevalier, F., Grossman, T., and Fitzmaurice, G., “Kitty: Sketching Dynamic and Interactive Illustrations,” *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST ’14, ACM, New York, NY, USA, 2014, pp. 395–405.
- [220] Davis, R. C., Saponas, T. S., Shilman, M., and Landay, J. A., “SketchWizard: Wizard of Oz Prototyping of Pen-based User Interfaces,” *User Interface Software and Technology*, UIST, 2007, pp. 119–128.
- [221] Molin, L., “Wizard-of-Oz Prototyping for Co-operative Interaction Design of Graphical User Interfaces,” *Proceedings of the Third Nordic Conference on Human-computer Interaction*, NordiCHI ’04, ACM, New York, NY, USA, 2004, pp. 425–428.
- [222] Greenberg, S., Carpendale, S., Marquardt, N., and Buxton, B., *Sketching user experiences: The workbook*, Elsevier, 2011.
- [223] Myers, B. A., McDaniel, R., and Wolber, D., “Programming by Example: Intelligence in Demonstrational Interfaces,” *Commun. ACM*, Vol. 43, No. 3, March 2000, pp. 82–89.
- [224] Nardi, B. A., *A small matter of programming: perspectives on end user computing*, MIT press, 1993.
- [225] Myers, B. A., Ko, A. J., and Burnett, M. M., “Invited Research Overview: End-user Programming,” *CHI ’06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA ’06, ACM, New York, NY, USA, 2006, pp. 75–80.
- [226] Cypher, A., Halbert, D. C., Kurlander, D., Lieberman, H., Maulsby, D., Myers, B. A., and Turransky, A., editors, *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, USA, 1993.
- [227] Lee, S. W., Freeman, J., Colella, A., Yao, S., and Van Troyer, A., “Evaluating Collaborative Laptop Improvisation with LOLC,” *Proceedings of the Symposium on Laptop Ensembles and Orchestras*, 2012, pp. 55–62.
- [228] MacKay, M., “Method Draw,” <https://github.com/duopixel/Method-Draw>, 2017.

- [229] Guo, P. J., “Online Python Tutor: Embeddable Web-based Program Visualization for Cs Education,” *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, SIGCSE ’13*, ACM, New York, NY, USA, 2013, pp. 579–584.
- [230] Victor, B., “Inventing on principle,” 2012.
- [231] Lee, S. W. and Essl, G., “Live Coding the Audience Participation,” *Proceedings of International Conference on Live Coding*, Hamilton, ON, Canada, 2016.
- [232] Riley, T., “In C,” *Composition*, 1964.
- [233] Wessel, D. and Wright, M., “Problems and prospects for intimate musical control of computers,” *Computer Music Journal*, Vol. 26, No. 3, 2002, pp. 11–22.
- [234] Gurevich, M., Stapleton, P., and Marquez-Borbon, A., “Style and constraint in electronic musical instruments,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Vol. 10, 2010.
- [235] Kirk, R. E., *Experimental design*, Wiley Online Library, 1982.
- [236] Overflow, S., “Stack Overflow, <https://stackoverflow.com/>,” 2015, Accessed: April, 2016.
- [237] LaToza, T. D., Venolia, G., and DeLine, R., “Maintaining mental models: a study of developer work habits,” *Proceedings of the 28th international conference on Software engineering*, ACM, 2006, pp. 492–501.
- [238] Herbsleb, J. D., Klein, H., Olson, G. M., Brunner, H., Olson, J. S., and Harding, J., “Object-oriented analysis and design in software project teams,” *Human–Computer Interaction*, Vol. 10, No. 2-3, 1995, pp. 249–292.
- [239] Raymond, E. S., *The Cathedral and the Bazaar*, O’Reilly & Associates, Inc., Sebastopol, CA, USA, 1st ed., 1999.
- [240] Ackerman, M. S., “Augmenting organizational memory: a field study of answer garden,” *ACM Transactions on Information Systems (TOIS)*, Vol. 16, No. 3, 1998, pp. 203–224.
- [241] Ackerman, M. S. and McDonald, D. W., “Answer Garden 2: merging organizational memory with collaborative help,” *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, ACM, 1996, pp. 97–105.
- [242] Ackerman, M. S. and Palen, L., “The Zephyr Help Instance: promoting ongoing activity in a CSCW system,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 1996, pp. 268–275.
- [243] Liu, X., Croft, W. B., and Koll, M., “Finding experts in community-based question-answering services,” *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, 2005, pp. 315–316.

- [244] Zhang, J., Ackerman, M. S., and Adamic, L., “Expertise networks in online communities: structure and algorithms,” *Proceedings of the 16th international conference on World Wide Web*, ACM, 2007, pp. 221–230.
- [245] Zhang, J., Ackerman, M. S., Adamic, L., and Nam, K. K., “QuME: a mechanism to support expertise finding in online help-seeking communities,” *Proceedings of the 20th annual ACM symposium on User interface software and technology*, ACM, 2007, pp. 111–114.
- [246] Riahi, F., Zolaktaf, Z., Shafiei, M., and Milios, E., “Finding expert users in community question answering,” *Proceedings of the 21st international conference companion on World Wide Web*, ACM, 2012, pp. 791–798.
- [247] Jeon, J., Croft, W. B., and Lee, J. H., “Finding semantically similar questions based on their answers,” *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2005, pp. 617–618.
- [248] Cao, Y., Duan, H., Lin, C.-Y., Yu, Y., and Hon, H.-W., “Recommending questions using the mdl-based tree cut model,” *Proceedings of the 17th international conference on World Wide Web*, ACM, 2008, pp. 81–90.
- [249] Asaduzzaman, M., Mashiyat, A. S., Roy, C. K., and Schneider, K. A., “Answering questions about unanswered questions of stack overflow,” *Proceedings of the 10th Working Conference on Mining Software Repositories*, IEEE Press, 2013, pp. 97–100.
- [250] Cockburn, A. and Williams, L., “The costs and benefits of pair programming,” *Extreme programming examined*, 2000, pp. 223–247.
- [251] Baheti, P., Gehringer, E., and Stotts, D., “Exploring the efficacy of distributed pair programming,” *Extreme Programming and Agile Methods—XP/Agile Universe 2002*, Springer, 2002, pp. 208–220.
- [252] Schenk, J., Prechelt, L., and Salinger, S., “Distributed-Pair Programming can work well and is not just Distributed Pair-Programming,” *Companion Proceedings of the 36th International Conference on Software Engineering*, ACM, 2014, pp. 74–83.
- [253] Olson, G. M. and Olson, J. S., “Distance Matters,” *Human-computer interaction*, Vol. 15, No. 2, 2000, pp. 139–178.
- [254] Sillito, J., Murphy, G. C., and De Volder, K., “Asking and answering questions during a programming change task,” *Software Engineering, IEEE Transactions on*, Vol. 34, No. 4, 2008, pp. 434–451.
- [255] Ko, A. J., Myers, B., Aung, H. H., et al., “Six learning barriers in end-user programming systems,” *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, IEEE, 2004, pp. 199–206.

- [256] Guzzi, A., Bacchelli, A., Riche, Y., and van Deursen, A., “Supporting Developers’ Coordination in the IDE,” *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ACM, 2015, pp. 518–532.
- [257] Guo, P. J., “Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming,” *Proceedings of the 28th annual ACM symposium on User interface software and technology*, ACM, 2015.
- [258] Guo, P. J., White, J., and Zanelatto, R., “Codechella: Multi-User Program Visualizations for Real-Time Tutoring and Collaborative Learning,” *Visual Languages and Human-Centric Computing (VL/HCC), 2015 IEEE Symposium on*, IEEE, 2015.
- [259] Steinmacher, I., Silva, M. A. G., and Gerosa, M. A., “Barriers faced by newcomers to open source projects: a systematic review,” *Open Source Software: Mobile Open Source Technologies*, Springer, 2014, pp. 153–163.
- [260] Robillard, M. P., Walker, R. J., and Zimmermann, T., “Recommendation systems for software engineering,” *Software, IEEE*, Vol. 27, No. 4, 2010, pp. 80–86.
- [261] Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., and Klemmer, S. R., “Two studies of opportunistic programming: interleaving web foraging, learning, and writing code,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2009, pp. 1589–1598.
- [262] Brandt, J., Guo, P. J., Lewenstein, J., Klemmer, S. R., and Dontcheva, M., “Writing Code to Prototype, Ideate, and Discover,” *Software, IEEE*, Vol. 26, No. 5, 2009, pp. 18–24.
- [263] Brandt, J., Dontcheva, M., Weskamp, M., and Klemmer, S. R., “Example-centric programming: integrating web search into the development environment,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 513–522.
- [264] Ponzanelli, L., Bacchelli, A., and Lanza, M., “Seahawk: Stack overflow in the IDE,” *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 2013, pp. 1295–1298.
- [265] Hartmann, B., MacDougall, D., Brandt, J., and Klemmer, S. R., “What would other programmers do: suggesting solutions to error messages,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 1019–1028.
- [266] Lasecki, W., Miller, C., Sadilek, A., Abumoussa, A., Borrello, D., Kushalnagar, R., and Bigham, J., “Real-time captioning by groups of non-experts,” *Proceedings of the 25th annual ACM symposium on User interface software and technology*, ACM, 2012, pp. 23–34.
- [267] “Upwork Inc. (formerly oDesk), <https://www.upwork.com>,” 2015, Accessed: April, 2016.

- [268] Chong, J. and Hurlbutt, T., “The social dynamics of pair programming,” *29th International Conference on Software Engineering (ICSE’07)*, IEEE, 2007, pp. 354–363.
- [269] Almaatouq, A., Alhasoun, F., Campari, R., and Alfaris, A., “The influence of social norms on synchronous versus asynchronous communication technologies,” *Proceedings of the 1st ACM international workshop on Personal data meets distributed multimedia*, ACM, 2013, pp. 39–42.
- [270] Card, S. K., Moran, T. P., and Newell, A., “The keystroke-level model for user performance time with interactive systems,” *Communications of the ACM*, Vol. 23, No. 7, 1980, pp. 396–410.
- [271] Bälter, O., “Keystroke level analysis of email message organization,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2000, pp. 105–112.
- [272] Joyce, R. and Gupta, G., “Identity authentication based on keystroke latencies,” *Communications of the ACM*, Vol. 33, No. 2, 1990, pp. 168–176.
- [273] Monroe, F. and Rubin, A. D., “Keystroke dynamics as a biometric for authentication,” *Future Generation computer systems*, Vol. 16, No. 4, 2000, pp. 351–359.
- [274] Vizer, L. M., Zhou, L., and Sears, A., “Automated stress detection using keystroke and linguistic features: An exploratory study,” *International Journal of Human-Computer Studies*, Vol. 67, No. 10, 2009, pp. 870–886.
- [275] Epp, C., Lippold, M., and Mandryk, R. L., “Identifying emotional states using keystroke dynamics,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2011, pp. 715–724.
- [276] Shukla, P. and Solanki, R., “Web Based Keystroke Dynamics Application for Identifying Emotional State,” .
- [277] Salmeron-Majadas, S., Santos, O. C., and Boticario, J. G., “An evaluation of mouse and keyboard interaction indicators towards non-intrusive and low cost affective modeling in an educational context,” *Procedia Computer Science*, Vol. 35, 2014, pp. 691–700.
- [278] Lee, J. C., Forlizzi, J., and Hudson, S. E., “The kinetic typography engine: an extensible system for animating expressive text,” *Proceedings of the 15th annual ACM symposium on User interface software and technology*, ACM, 2002, pp. 81–90.
- [279] Adamczyk, P. D. and Bailey, B. P., “If not now, when?: the effects of interruption at different moments within task execution,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2004, pp. 271–278.
- [280] Baudisch, P., Tan, D., Collomb, M., Robbins, D., Hinckley, K., Agrawala, M., Zhao, S., and Ramos, G., “Phosphor: Explaining Transitions in the User Interface Using Afterglow Effects,” *Proceedings of the 19th Annual ACM Symposium on User*

Interface Software and Technology, UIST '06, ACM, New York, NY, USA, 2006, pp. 169–178.

- [281] Larson, E. and von Eye, A., “Predicting the perceived flow of time from qualities of activity and depth of engagement,” *Ecological Psychology*, Vol. 18, No. 2, 2006, pp. 113–130.
- [282] Lee, S. W. and Essl, G., “Models and Opportunities for Networked Live Coding,” *Proceedings of The Live Coding and Collaboration symposium 2014*, Birmingham, United Kingdom, 2014.
- [283] Lee, S. W., Bang, J., and Essl, G., “Live Coding YouTube: Organizing Streaming Media for an Audiovisual Performance,” *Proceedings of the International Conference on New Interfaces for Musical Expression*, Aalborg University Copenhagen, Copenhagen, Denmark, 2017, pp. 261–266.
- [284] Lim, S. C., *Realizing empathy: An inquiry into the meaning of making*, Seung Chan Lim, 2013.
- [285] Decety, J. and Lamm, C., “Human empathy through the lens of social neuroscience,” *The scientific World journal*, Vol. 6, 2006, pp. 1146–1163.
- [286] Hollan, D. and Throop, C. J., “Whatever happened to empathy?: Introduction,” *Ethos*, Vol. 36, No. 4, 2008, pp. 385–401.