VizPI: A Real-Time Visualization Tool for Enhancing Peer Instruction in Large-Scale Programming Lectures

Xiaohang Tang
Xi Chen
Virginia Tech
USA
{xiaohangtang,xic}@vt.edu

Sam Wong
University of California, San Diego
USA
c6wong@ucsd.edu

Yan Chen
Virginia Tech
USA
ych@vt.edu

ABSTRACT

Peer instruction (PI) has shown significant potential in facilitating student engagement and collaborative learning. However, the implementation of PI for large-scale programming lectures has proven challenging due to difficulties in monitoring student engagement, discussion topics, and code changes. This paper introduces VizPI, an interactive web tool that enables instructors to conduct, monitor, and assess PI for programming exercises in real-time. With features that visualize the progress of student discussions and code submissions, VizPI allows for more effective oversight of PI activities and the provision of personalized feedback at scale. Our work aims to transform the pedagogical approach to PI in programming education, making it more engaging and adaptable to student needs.

ACM Reference Format:
Xiaohang Tang, Xi Chen, Sam Wong, and Yan Chen. 2023. VizPI: A Real-Time Visualization Tool for Enhancing Peer Instruction in Large-Scale Programming Lectures. In *The 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23 Adjunct), October 29–November 01, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3586182.3616632

1 INTRODUCTION

Peer Instruction (PI) is an instructional strategy developed by Mazur et al. that emphasizes students' active construction of conceptual understanding [1]. Initially employed in physics lectures for multiple-choice questions, students first individually respond to a question (the individual vote using wireless keypads or iclickers), then engage in peer discussion, and finally cast a re-vote influenced by the discussion (the group vote). Many recent studies have reported the benefits of PI in computing education [2] with students' survey responses overwhelmingly endorsing the efficacy of PI for learning and recommending its continued usage in further courses. Instructors, too, benefit from a refined focus on student difficulties, an improved ability to adapt lectures in real-time, and enhanced student involvement in teamwork and collaboration [4].

Recently, with the advent of supporting tools [3, 6], programming instructors have begun using in-class coding exercises — small scale programming exercises for students to practice during lectures or lab — as the basis for PI activities [5]. Instead of limiting discussions to options with immediate neighbors in a classroom, the use of online tools has enabled broader student interaction, facilitating code sharing and collaborative problem-solving. Despite these advancements, our preliminary interviews and previous research indicate that instructors struggle to assess student engagement during peer discussions, understand the topics being discussed, or ensure the relevance of the discussions to the exercise at hand. This becomes especially challenging in introductory programming classes, often comprising hundreds of students. Therefore, our research question arose: how can we aid instructors in effectively monitoring PI, specifically peer discussion activities, for coding exercises in large-scale programming lectures?

This poster presents our ongoing work aimed at enhancing the efficacy of PI in programming lectures. We introduce a prototype, VizPI[1], an interactive web tool that enables instructors to monitor numerous group discussions in real-time. With VizPI, instructors can create a coding exercise for students, while the tool monitors the submission rate and displays it to the instructor. Students are then grouped in pairs or trios, with a balanced expertise within each group. VizPI provides real-time visualization of each group's discussion progress and success rate. This allows instructors not only to gauge the overall progress of group discussions but also to discern common discussion points, prevalent mistakes, and other salient features of student discourse. As a result, instructors can easily highlight important codes or issues that VizPI has identified, creating a lecture that is considerably more adaptive to ongoing discussions. This innovative tool aids instructors in providing personalized, scalable feedback within the lecture setting.

2 INFORMAL NEEDFINDING INTERVIEWS

We conducted informal interviews with three instructors from our institution who teach introductory programming classes, gathering insights about their needs concerning peer instruction. These insights have significantly informed our system design rationale: (1) Instructors expressed a desire to comprehend and discuss overarching group performance metrics, particularly the primary mental, coding, and logical challenges that students discuss. (2) They also emphasized the utility of being able to present exemplary student responses on a large screen for class-wide discussion. This approach inverts the traditional pedagogical model, wherein instructors primarily share examples. (3) In addition, instructors highlighted the potential for innovation when our tool is integrated with a well-designed pedagogical strategy for classroom use. Such a blend of

[1]Short for visualization for peer instructions.

UIST '23 Adjunct, October 29–November 01, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0096-5/23/10.
https://doi.org/10.1145/3586182.3616632

The Need for Real-Time Crowd Generation of Task Lists from Speech

Sang Won Lee[1], **Yan Chen**[2], **Walter S. Lasecki**[1,2]
Computer Science & Engineering[1] School of Information[2]
MISC Group MISC Group
University of Michigan, Ann Arbor University of Michigan, Ann Arbor

{snaglee,yanchenm,wlasecki}@umich.edu

Introduction

Crowdsourcing has made robust speech-based interactive systems possible (Lasecki et al. 2012; 2014; 2015). Speech-based interaction provides an effective and immediate way for workers to understand requester intent via streaming audio (Lasecki et al. 2015; Lee et al. 2017), or in near real-time via audio recordings (Bigham et al. 2010; Nebeling et al. 2016; Chen et al. 2017; Zhong et al. 2015). Speech is a powerful way to describe requests because of the flexibility and context that can be easily provided (e.g., *Can you recommend a birthday present for a five year old boy? I don't know what kids like these days. I don't want to spend more than $30. It's for my nephew.*).

However, since the verbal description of a complex request can get lengthy, crowd workers may need to revisit it (given a recording), or ask the requester to repeat the description (given streaming audio). To that end, it would be beneficial for crowd workers to organize and archive complex requests for later navigation and retrieval information. In this paper, we introduce *Speech-To-Tasks*, a tool that takes audio input and generates a hierarchical list of sub-tasks.

Our *Speech-To-Tasks* system leverages crowds to segment audio into multiple parts and presents them hierarchically so that a complex task can be decomposed into a set of subtasks. The generated list of items can be used for crowd workers to self-coordinate, to avoid conflicts, as well as for requesters to monitor their progress. Further, converting speech into a structured summary itself can be useful in various settings (e.g., taking notes in meetings, summarizing a lecture, generating to-do items from a verbal request). We aim to provide a general solution for speech-based tasks, especially for near/real-time applications in crowdsourcing (Lasecki et al. 2011). In addition, findings from this work can help inform the design of intelligent conversational assistants like Apple Siri or Amazon Echo, and improve such systems to resolve more complex tasks.

In this paper, we outline the motivation and challenges in making complex verbal requests, and introduce our ongoing work with the following target contributions:

- The idea of converting complex request in speech into a set of subtasks in a crowdsourcing system.

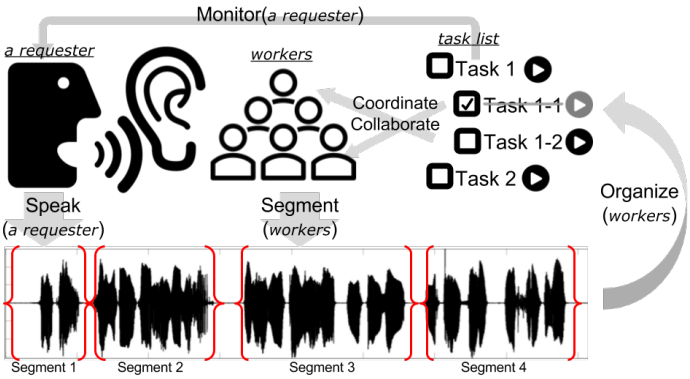

Figure 1: *Speech-To-Tasks* Process: Crowd workers segment audio into multiple parts and assign each segment to a hierarchical list. The list can be used for crowd workers to self-coordinate in case of real-time collaboration and for the requester to monitor the progress of the tasks.

- Identifying challenges and costs in verbally describing complex tasks.
- Methods and tools that help crowd workers segment speech input and synthesize a list of hierarchical subtasks.

Challenges in Describing Complex Tasks

Speech Recognition vs. Human Computation Defining tasks using text-based topic modeling or automatic speech recognition (ASR) is challenging due to error-prone results of ASR, the difficulty in understanding the intent of natural language, or missing context (Yamamoto, Ogata, and Ariki 2003; Glass et al. 2007; Repp, Grob, and Meinel 2008). Even with significant advancements in the accuracy of ASR, the consequences of errors can be costly in real-world applications and using automated methods to extract semantic relationships between audio segments remains a harder, farther-off goal. Instead, we combine human computation and simple audio processing techniques in this work in order to manually segment audio into meaningful subtasks. We then ask crowd workers to briefly annotate the audio segment or associate it with relevant elements in the task. The output of the system can be used in coordinating real-time collaboration between crowd workers once generated within the context that the list is generated.

Towards Software Development Microtasks

Yan Chen
School of Information
University of Michigan, Ann Arbor
yanchenm@umich.edu

Steve Oney
School of Information
University of Michigan, Ann Arbor
soney@umich.edu

Walter S. Lasecki
Computer Science & Engineering
and School of Information
University of Michigan, Ann Arbor
wlasecki@umich.edu

Abstract
Software development is a large, complex task that is difficult to break down into simple pieces that can be efficiently integrated into a single codebase. Developers often decompose programing tasks into classes, functions, and other constructs based on their functionality. However, solving programming tasks often requires adequate contextual information and efficient ways to aggregate the responses. In this paper, we discuss ongoing work on a system that helps developers easily (e.g. informally saying it) generate well-defined programming microtasks by enabling multimodal interactions. These tasks are flexible, schedulable, and their solutions can be efficiently integrated into the larger codebase.

Capturing Developers' Context
In prior studies, we have found that programming tasks generated during the software development process require significant added contextual information to be understood [1]. Previous work also suggested that creating tasks with enough context captured can decrease the time spent on recovery from interruption [2, 3]. Ideally, recovery time should be reduced as much as possible. This leads to our question: *How can a system automatically capture or generate task context that will minimize developers' effort?*

Mocking-up Desired UI Behaviors from UI Element-Based Recording

Yan Chen
yanchenm@umich.edu || *University of Michigan, Ann Arbor*

I. Introduction

Software developers often ask for support from other developers, but effective communication about programming problems can be challenging. In the context of user interface (UI) development, effective communication about interactive behaviors of a UI is particularly difficult as it often requires a visual demonstration of the UI behaviors as supporting context. My motivational study found that our participants can always correctly understand a request when it includes video demos of the problem UI behavior and desired UI behavior. I summarized that an ideal request regarding a UI interactive behavior problem should include interrelated natural language description, relevant code, and demonstrations of non-desired and desired UI behaviors. I observed that developers often provide only the demonstration of not desired UI behavior and then describing the desired behavior on top of it. Unable to provide desired UI behaviors makes communication about UI behavior ineffective, and I argue this is a limitation of existing techniques. In this work, I would like to propose a solution to address it.

Existing tools, such as online discussion forums, allow developers to take multiple screenshots or insert examples via external links (e.g., jsFiddle.com) to provide visual context. However, capturing these behaviors often requires a sequence of user inputs which can be difficult to express in a static and linear form (e.g., text or screenshots) that is easy to understand. Prior work has argued that online discussion forums should be domain-specifically designed for more effectively communication [1]. Video recording these behaviors could be a solution, but requesters might miss necessary contextual information in their description. For example, our preliminary study found that when describing a desired behavior for one UI element, participants often forgot to provide the constraints for how other UI elements' should behave. Video recordings could support better communication of ideas with appropriate visual context, but the content in videos (e.g., UI elements) are not reusable.

II. Motivational Studies

To find more evidence of this problem, I conducted an in-lab exploratory study with seven undergrads from my university. Each session lasts for an hour. All of them had at least one year of experience developing interactive UIs and had completed a team project in their UI development class. I asked three of them to act as a requester and make requests about a set of common UI behavior problems found on a web UI course (UMich EECS493) using a commercial tool, Scrimba [2]. We used Scrimba as it captures users' keystroke level activities and cursor movement on its editor, which provides more information in the recording. Participants were provided code, incorrect output, and the desired output. I asked the remaining four participants to act as helpers, to comprehend these requests and describe what they understood about the request. I then showed them the problems that were presented to the requesters and ask them to evaluate the requests by comparing their understanding to the ground-truth. In summary, I found that

- helpers can correctly understand all the requests when *both* a natural language description and video demos of incorrect and desired UI behaivor outputs were given,
- without being able to record the desired behaviors, requesters would often forget to describe some information in the requests,
- without videos of the desired behaviors, helpers would inaccurately understand the requests.

Based on these findings, I am proposing a new system to make it easier to generate desired UI behaviors. One idea that

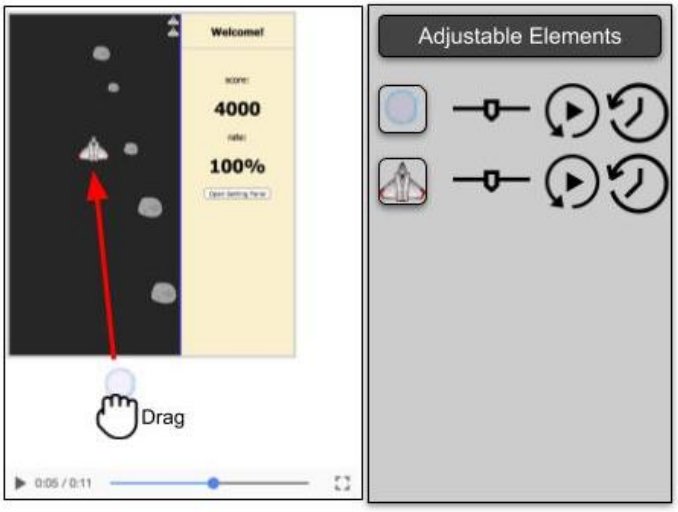

Fig. 1. CoCapture user interface. On the left is the recorded demonstration of non-desired UI behaviors. On the right is a list of DOM elements that are recorded independently. As each element is recorded as its own demonstration, developers can edit their properties, like speed, or location, independently, and remix the elements to reconfigure the desired UI behaviors.

978-1-7281-0810-0/19/$31.00 ©2019 IEEE

Expert Crowd Support Systems for Software Developers

YAN CHEN, STEVE ONEY, WALTER S. LASECKI, University of Michigan, Ann Arbor

1. INTRODUCTION

Software programming requires strong logical reasoning, proficiency in programming syntax, and a great deal of coding experience. In order to make software development more efficient, developers often seek programming support from various resources, such as the Web and other programmers [Hartmann et al. 2010]. However, our previous studies have shown that current support tools are limited for many of the types of requests developers would like to make, such as high-level advice, personalized help, or project-specific code segments [Chen et al. 2016].

Current IDE tools like Blueprint [Brandt et al. 2010] provide contextualized, easily-accessible, and on-demand support for developers, but are generally limited in the types of feedback they can provide (e.g., syntax error highlighting and function auto-complete) because the system cannot truly understand user queries or the context of the problem. Recruiting a programming helper can provide personalized support for specific requests, real time interactions and feedback, and even the ability to hand off sub-tasks for independent completion. However, in addition to the monetary price of hiring a freelancer, current hiring processes take significant time and preparation effort costs (interviews, onboarding, etc.). Furthermore, none of these helper solutions provide the in-context support that IDEs do, adding the need for an additional context switch to and from a developers workflow.

To overcome these limitations, we propose a new class of support systems for software developers to request on-demand help by expert crowds. Unlike automated IDE tools, expert crowds can provide high-level feedback about code by inferring and considering the end-user developer's intent. Unlike developer forums, on-demand developers can provide rapid responses and write project-specific code segments (which is often explicitly discouraged in developer forums). This paper describes three studies that help inform the design of on-demand developer support systems. We then discuss the implications draw from the results and conclude with design guidelines for future systems.

2. EXPLORATORY STUDIES

We conducted studies to inform design of each of the three main stages of help seeking: users forming a request, the worker response process, and users integrating responses. To minimize the affects of prior expertise between subjects, our studies used a new programming language that we generated from parts of multiple existing languages. In this section, we describe each of our studies.

2.1 Study 1: Developers' Input Methods

To find out what methods allow developers to make requests easily and quickly, we compared three request modalities for describing requests: 1) speak the request (Voice), 2) type the request (Text), and 3) choose the request from a fixed set of options (Multiple Choice). We also compared two context selectors for specifying a request's context: 1) select a segment of content (Highlight), and 2) point to one location in the content (Click). The multiple choice options are pre-selected from the types of request that we found in our previous studies [Chen et al. 2016], and the multiple location options are the most common programming language constructors.

Exploring Coordination Models for Ad Hoc Programming Teams

Sang Won Lee
Computer Science & Eng.
University of Michigan
snaglee@umich.edu

Sai R. Gouravajhala
Computer Science & Eng.
University of Michigan
sairohit@umich.edu

Yan Chen
School of Information
University of Michigan
yanchenm@umich.edu

Angela Chen
Computer Science & Eng.
University of Michigan
chenaj@umich.edu

Noah Klugman
Computer Science & Eng.
University of Michigan
nklugman@umich.edu

Walter S. Lasecki
Computer Science & Eng.
University of Michigan
wlasecki@umich.edu

CHI'17 Extended Abstracts, May 06-11, 2017, Denver, CO, USA
ACM 978-1-4503-4656-6/17/05.
http://dx.doi.org/10.1145/3027063.3053268

Abstract
Software development is a complex task with inherently interdependent sub-components. Prior work on crowdsourcing software engineering has addressed this problem by performing an a priori decomposition of the task into well-defined microtasks that individual crowd workers can complete independently. Alternatively, ad hoc teams of experts recruited from online crowds can remotely collaborate, avoiding the up-front cost to end users of task decomposition. However, these temporary ad hoc teams can lead to high coordination costs during the session itself. In this paper, we explore the types and causes of these coordination costs for transient software teams in existing collaborative programming tools: a version control system and a real-time shared editor. Based on our findings, we suggest design elements of shared programming environments that help teams effectively self-coordinate on their task.

Author Keywords
Software development tools; Ad hoc teams; Crowdsourcing

ACM Classification Keywords
H.5.m [Information interfaces and presentation]: Misc.

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 pro-

Toward a Non-Intrusive, Physio-Behavioral Biometric for Smartphones

Esther Vasiete
Department of Computer Science
University of Colorado Boulder
esther.vasiete@colorado.edu

Yan Chen
Department of Computer Science
University of Colorado Boulder
yan.chen@colorado.edu

Ian Char
Department of Computer Science
University of Colorado Boulder
ian.char@colorado.edu

Tom Yeh
Department of Computer Science
University of Colorado Boulder
tom.yeh@colorado.edu

Vishal Patel
UMIACS
University of Maryland
College Park
pvishalm@umiacs.umd.edu

Larry Davis
UMIACS
University of Maryland
College Park
lsd@umiacs.umd.edu

Rama Chellappa
UMIACS
University of Maryland
College Park
rama@umiacs.umd.edu

MobileHCI '14, Sep 23–26 2014, Toronto, ON, Canada
ACM 978-1-4503-3004-6/14/09.
http://dx.doi.org/10.1145/2628363.2634223

Abstract
Biometric authentication relies on an individual's inner characteristics and traits. We propose an active authentication system on a mobile device that relies on two biometric modalities: 3D gestures and face recognition. The novelty of our approach is to combine 3D gesture and face recognition in a non-intrusive and unconstrained environment; the active authentication system is running in the background while the user is performing his/her main task.

Author Keywords
3D gesture recognition; behavioral biometric authentication; face recognition; fusion system.

ACM Classification Keywords
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

Introduction
The most common user identification techniques on mobile devices are based on passwords or on graphical puzzles. However, there are three shortcomings with these techniques. First, passwords in memory-based approaches can be forgotten, stolen, or hacked. Second, after the initial login there is no further identification procedure as long as the device remains active. An impostor could gain physical access to the device if the legitimate user leaves it unlocked and unattended. Third, the authentication mechanism is