Codeopticon: Real-Time, One-To-Many Human Tutoring for Computer Programming

Philip J. Guo
University of Rochester
Rochester, New York, USA
pg@cs.rochester.edu

ABSTRACT

One-on-one tutoring from a human expert is an effective way for novices to overcome learning barriers in complex domains such as computer programming. But there are usually far fewer experts than learners. To enable a single expert to help more learners at once, we built Codeopticon, an interface that enables a programming tutor to monitor and chat with dozens of learners in real time. Each learner codes in a workspace that consists of an editor, compiler, and visual debugger. The tutor sees a real-time view of each learner's actions on a dashboard, with each learner's workspace summarized in a tile. At a glance, the tutor can see how learners are editing and debugging their code, and what errors they are encountering. The dashboard automatically reshuffles tiles so that the most active learners are always in the tutor's main field of view. When the tutor sees that a particular learner needs help, they can open an embedded chat window to start a one-on-one conversation. A user study showed that 8 first-time Codeopticon users successfully tutored anonymous learners from 54 countries in a naturalistic online setting. On average, in a 30-minute session, each tutor monitored 226 learners, started 12 conversations, exchanged 47 chats, and helped 2.4 learners.

Author Keywords

learning at scale; computer programming; remote tutoring

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation (e.g. HCI): Miscellaneous

INTRODUCTION

Decades of computing education research has shown that programming is hard to learn alone [11, 19]. Novices often struggle to develop robust mental models of code execution and are thus susceptible to hundreds of common misconceptions about how their code works [19]. One-on-one tutoring from a human expert is one of the most effective ways for novices to overcome these common learning barriers [3]. This format is powerful because a tutor can sit beside a learner who is debugging their code, observe their actions firsthand, and then provide *timely*, *targeted*, and *proactive* help when needed [3].

However, this kind of high-touch interaction does not scale, since there are far fewer tutors than learners. MOOCs enroll up to tens of thousands of learners, and many university courses are also growing large. For instance, introductory computer science courses at UC Berkeley and the University of Washington enroll over one thousand students per term [1]. Even with dozens of teaching assistants (TAs), there is still at least an order of magnitude more students than course staff.

One practical way to amplify human expertise in these settings is via asynchronous interfaces such as discussion forums, mailing lists, and Q&A sites. Although these interfaces are scalable and easy to deploy, they have some limitations: Novices often have a hard time phrasing their questions properly and thus may not get any useful responses [2], lose mental context while waiting for responses to arrive [12], and may not know when to step back and even ask a question when they are fixated on debugging a specific code error [21].

Instead, what if an expert could remotely monitor multiple learners as they are coding, see who seems to be struggling, and then jump in to offer timely, targeted, and proactive help?

In this paper, we present *Codeopticon*, a prototype real-time, one-to-many tutoring interface that explores this question. Codeopticon embodies a novel synchronous approach to providing expert help for computer programming, which complements existing asynchronous approaches such as discussion forums. Figure 1 shows its main features:

- Each learner works in an online workspace that consists of a code editor, compiler, and visual debugger. These sorts of coding workspaces are common in online learning environments such as MOOCs and Khan Academy.
- The tutor sees a real-time view of each learner's actions on a dashboard, with each learner's workspace state represented in a tile. At a glance, the tutor can see how learners are editing and debugging their code, and what errors they are encountering. The dashboard automatically reshuffles tiles based on levels of learner activity so that the most active learners are always in the tutor's main field of view.
- When the tutor sees that a learner needs help, they can open an embedded chat window in the appropriate tile to start a one-on-one, text-based chat with that learner.

UIST '15, November 08–11, 2015, Charlotte, NC, USA

ACM 978-1-4503-3779-3/15/11...$15.00
DOI: http://dx.doi.org/10.1145/2807442.2807469

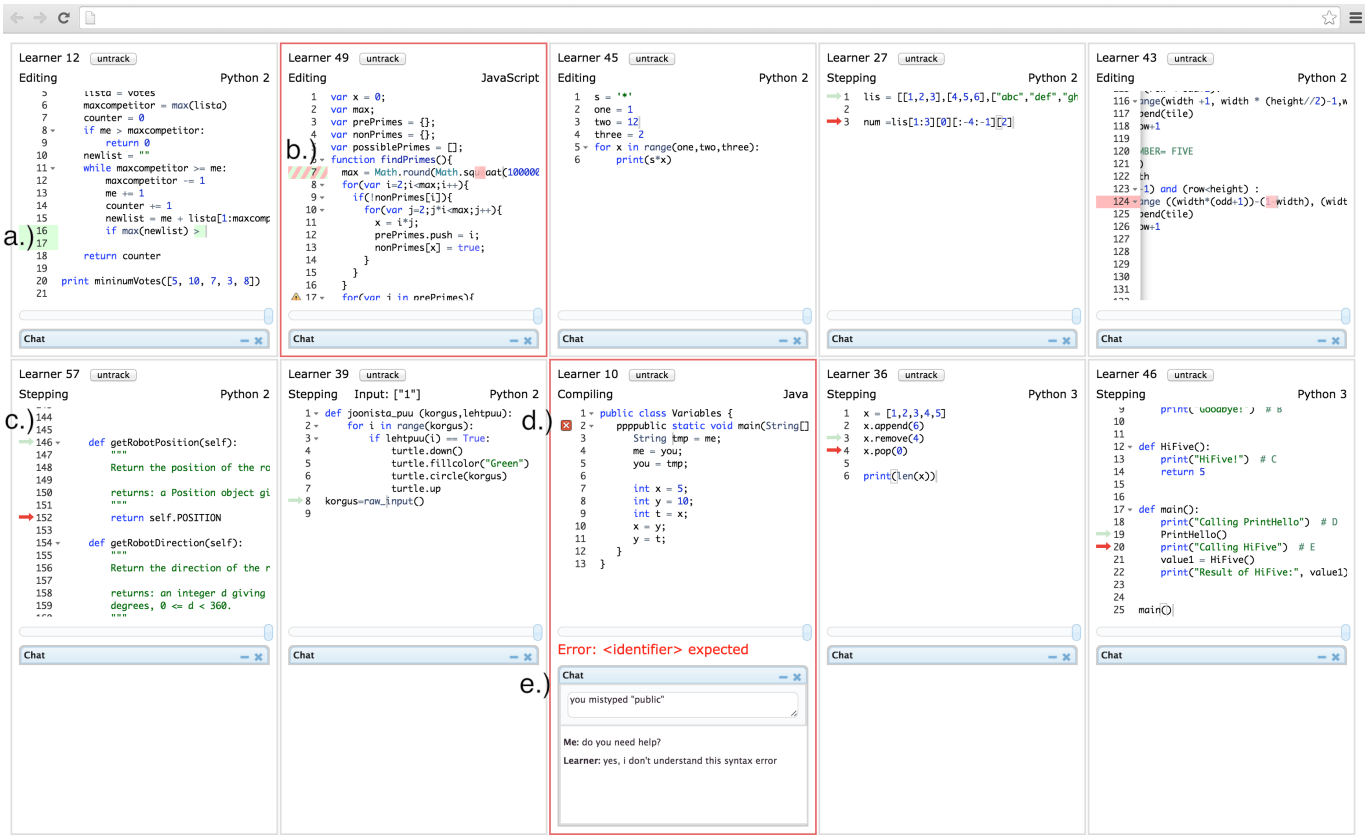

Figure 1. The Codeopticon interface consists of an array of tiles (10 shown here), each displaying one particular learner's code editing and execution actions in real time. Code edits are shown as diffs with a.) insertions in green and b.) deletions in red. c.) Arrows point to the current and next lines when the learner is stepping through executed code in a debugger. d.) Error messages are shown in red. e.) Each tile contains an embedded chat box.

Codeopticon's design is inspired both by formative observations of tutors working in a university computer lab and by analysis of log data from over 65,000 learners using a popular online coding workspace [10]. The central design challenge is to create an interface that enables a single tutor to effectively monitor and help as many learners as possible in real time. It must also handle a heterogeneous population that is common in online environments, with up to dozens of learners each working at their own pace on different pieces of code, and entering and exiting the workspace at unpredictable times. To address this challenge, we developed two novel interface elements for Codeopticon: a real-time text diff visualization to display learner behavior, and dynamic tile reshuffling to hone the tutor's attention to the most active learners.

The main idea behind Codeopticon is that *building a one-to-many remote monitoring and chat interface will enable instructors to provide one-on-one tutoring services to more learners than they could feasibly handle in person.* Although Codeopticon is made for computer programming, some of its ideas could be adapted for scaling up tutoring in other educational domains such as writing or engineering design.

To validate this idea, we ran a user study where 8 teaching staff (1 instructor and 7 TAs) from an introductory programming course used Codeopticon to tutor anonymous learners who visited the Online Python Tutor [10] coding workspace.

Throughout a 30-minute session, on average each subject saw 226 learners, initiated 12 chat conversations, exchanged 47 messages, and helped 2.4 learners resolve their problems. Despite not knowing anything beforehand about who these learners were or what code they were writing, the subjects were still able to successfully tutor people in this naturalistic online setting. All subjects said that they found Codeopticon to be an authentic and compelling alternative to in-person tutoring in the computer lab, with advantages such as being able to gauge the collective progress of an entire class, to tutor multiple learners at once while controlling their own pace, and to discreetly and proactively reach out to shy learners.

The contributions of this paper are:

- The idea of building a one-to-many remote monitoring and chat interface, which enables instructors to provide valuable one-on-one tutoring services to more learners than they could feasibly handle in person.
- The Codeopticon system that embodies this idea for teaching computer programming, which incorporates novel interface elements such as real-time text diff visualization and dynamic tile reshuffling.
- Findings from a user study that demonstrates how Python instructors can use Codeopticon to successfully tutor anonymous learners in a naturalistic online setting.

RELATED WORK

Situational awareness tools provide an operator with a real-time view of a complex situation containing multiple actors and moving parts so that they can make informed decisions about it [18]. Examples include monitoring dashboards for air traffic controllers, military commanders, power plant operators, and emergency responders [20], and also collaborative workplace tools in domains such as writing [7]. Codeopticon strives to maximize a tutor's situational awareness when monitoring dozens of learners as they are coding. It borrows relevant techniques from prior work such as displaying the most salient data in the main field of view to minimize cognitive load, and providing a history slider to show past events.

CrowdScape [17] is an interactive visualization that allows requesters on Mechanical Turk to monitor crowd workers' activities to gauge the quality of their output. The motivating insight behind CrowdScape is that human judgment is needed to determine the quality of crowd work on open-ended, creative, and subjective sorts of tasks. Similarly, determining when a learner is struggling is an inherently subjective call that cannot easily be automated, so we designed Codeopticon to provide a tutor with the proper situational awareness to recognize when to intervene using their own best judgment.

In the educational domain, the two most closely related situational awareness tools are RIMES [13] and OverCode [8]. RIMES allows an instructor to embed interactive assignments within lecture videos. It aggregates all students' responses to each assignment in a dashboard for the instructor to give feedback and identify trends. OverCode aggregates thousands of student responses to a single computer programming assignment, clustering batches of similar code into virtual stacks. The instructor can grade and give feedback to an entire cluster at a time. Codeopticon was inspired by the idea of aggregating the activities of many learners into a compact dashboard. But neither RIMES nor OverCode are meant for providing real-time tutoring; instructors use them to view and comment on work offline. Also, they cluster multiple solutions to a *single* assignment, whereas Codeopticon enables a tutor to monitor learners working on their own independent assignments.

Researchers have also recently focused on social aspects of online education. As a complement to discussion forums, real-time chat improves motivation for learners in online settings by making them feel more connected to their peers [12]. However, Coetzee et al. found that generic chat rooms in MOOCs had low activity [5] and subsequently designed structured chat rooms where peers collaborated to solve specific exercises [6]. Talkabout [15] uses Google Hangouts to coordinate small-group peer video chats in MOOCs to discuss course material. Codeopticon was similarly inspired by the goal of humanizing online education by connecting people remotely via chat, but it uses chat for one-on-one tutoring rather than for group-based peer learning.

Finally, shared coding workspaces such as Collabode [9], CrowdCode [16], Cloud9 (`c9.io`), or even Google Docs enable people to write code together while text chatting. While these tools are meant for multiple people working on a single piece of code, Codeopticon is made for a single tutor to monitor multiple learners each writing their own unique pieces of code. We chose not to include a shared text editor since we did not want the tutor writing code on behalf of learners.

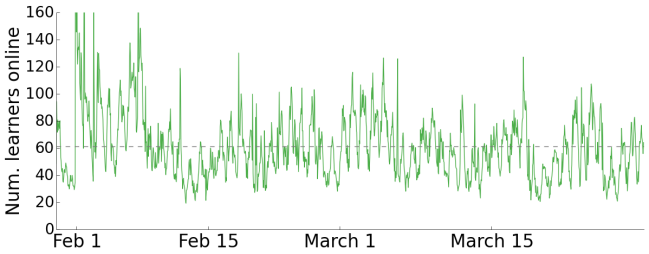

Figure 2. Number of learners concurrently online in the Online Python Tutor [10] coding workspace in Feb. and March 2015 (mean=61, σ=25)

To our knowledge, Codeopticon is the first attempt at a real-time, one-to-many monitoring and chat interface for scaling up human tutoring in the domain of computer programming.

FORMATIVE OBSERVATIONS AND DESIGN GOALS

To formulate design goals for Codeopticon, we observed teaching assistants (TAs) in our university's introductory programming course as they helped students in the computer lab. Each TA holds lab hours where they sit in the lab along with anywhere from ten to fifty students who are working on their coding assignments. When we interviewed seven TAs, they all expressed a strong desire to help more students at once without needing to physically run around the computer lab, which gets exhausting. Currently the only way to multiplex is to tell a student to try something, walk over to help another student, and then come back later to check on the first student. Also, some TAs were unsure of what to do when the lab was nearly silent, since they could not see what everyone was working on; maybe students were making steady progress, or maybe some were stuck and just reluctant to ask for help.

Although Codeopticon can be used in the classroom, our primary use case is in large-scale online education settings such as MOOCs, which contain a much larger and more heterogeneous learner population [14]. To quantify this heterogeneity, we analyzed two months of log data from Online Python Tutor, a Web-based coding workspace used by many MOOC participants and self-directed learners [10]. During Feb. and March 2015, ~67,520 learners from ~165 countries wrote and debugged their code in this workspace. Figure 2 shows that, on average, 61 learners were concurrently online at any moment, which means that Codeopticon must be able to monitor several dozen learners to be effective. There was also very high turnover, with a mean of 269 new learners arriving at (and leaving) the workspace each hour, and most sessions (site visits) being short: out of 447,608 sessions, 90% lasted for less than 10 minutes, 84% for less than 5 minutes, and 72% for less than 1 minute. Thus, Codeopticon must handle high variability in learner activity. Our three design goals are:

D1: Scale – A tutor should be able to remotely monitor up to several dozen learners at once. Since novices often have a hard time asking for help when fixated on a debugging task [21], the interface should enable the tutor to quickly rec-

ognize when a particular learner is stuck or thrashing, so that they can step in to offer proactive help via text chat.

D2: Multiplexing – A tutor should be able to help multiple learners at once without losing context, taking advantage of natural pauses in tutoring interactions when a learner is editing or executing code in response to the tutor's suggestions.

D3: Heterogeneity – Learners arrive and leave at unpredictable times, each work on their own independent pieces of code at their own pace, and have varying levels of expertise. Thus, Codeopticon cannot assume that all learners are progressing in lock step on a single programming assignment.

CODEOPTICON DESIGN AND IMPLEMENTATION

Codeopticon is a standard Web application that runs in any modern browser. It consists of two parts: 1.) augmenting an existing coding workspace with monitoring and embedded chat, and 2.) the main Codeopticon interface that a tutor uses.

Augmenting an Online Coding Workspace

To create Codeopticon, we first augmented an online coding workspace to log the learner's behavior. These workspaces consist of a Web-based code editor, compiler, and runtime system that executes the learner's code to produce outputs and error messages. Since they allow learners to write and debug code directly in the browser, coding workspaces have become a key part of computer programming MOOCs and courses from organizations such as Khan Academy and Codecademy.

We augmented Online Python Tutor (`pythontutor.com`), a standard workspace with one unique feature: a visual debugger that enables the learner to step forwards and backwards through execution and see a visualization of runtime state. Online Python Tutor (despite its name) supports five languages – Python, Java, JavaScript, TypeScript, and Ruby – and is widely used in MOOCs and university courses [10].

We added detailed logging and an embedded chat interface to Online Python Tutor. The left pane of Figure 3 walks through a typical learner interaction on the `pythontutor.com` website:

- Figure 3a. – Pick the language and write code. All code edits are logged as diffs and sent to the Codeopticon server.
- Figure 3b. – Compile the code and then use a slider or buttons to step forwards and backwards through execution points. All compilation and navigation actions are logged.
- Figure 3c. – Just like in a debugger, the current and next lines of code to execute are shown as arrows in the editor.
- Figure 3d. – See a visualization of stack frames, variables, objects, and pointers at the current execution point.
- Figure 3e. – See the program's `print` statement output.
- Figure 3f. – Chat with a tutor via a text-based chat widget.

The additions we made for Codeopticon are lightweight and can be easily implemented for other coding workspaces such as those in MOOCs, Khan Academy, or Codecademy.

In our current prototype, the learner cannot summon a tutor. However, if Codeopticon were integrated into, say, a MOOC platform, then the coding workspace could be augmented to show which TAs are currently holding virtual office hours.

Codeopticon Tutor Interface and Usage Scenario

The Codeopticon user interface (Figure 1) enables a tutor to simultaneously monitor and chat with multiple learners. It was implemented in HTML5, with real-time status updates and chat powered by WebSockets with `socket.io`.

Here is an example usage scenario: Alice is a teaching assistant (TA) for a computer programming MOOC. Before Codeopticon, the only practical way she could help learners was to monitor and reply to questions on the course discussion forum. She had no way of providing timely, targeted, and proactive help, and did not know how many learners were silently stuck without even knowing what to ask on the forum.

Using Codeopticon, Alice can sign on to see a dashboard of tiles (Figure 1), each visualizing one learner's actions. The right of Figure 3 zooms in on a single tile, which shows a real-time view of a learner's code editor (Figure 3a.), execution step arrows (Figure 3c.), and chat widget (Figure 3f.).

Alice cannot directly edit or execute anyone's code; she can only text chat with them. This interaction emulates the commonly-observed best practice of an in-person tutor sitting beside a learner and letting the learner do all of the typing.

Whenever a learner's code triggers a compile- or run-time error, a red X indicator appears next to the offending line, and the error message displays in bright red at the bottom of their tile (Figure 1d.). This visual saliency helps direct Alice's attention to learners who are struggling with code errors so that she can intervene to help right when the learner needs it most.

Even if a learner does not trigger an error, if Alice sees them thrashing back and forth executing similar pieces of code, then she might ping them to ask if they have questions. This proactive behavior mimics what conscientious TAs in the computer lab would do as they look over students' shoulders. Note that Codeopticon cannot automatically detect learner struggle; rather, it provides a means for a tutor to concurrently monitor many learners and use their own best judgment to determine who is possibly struggling.

Real-time Text Diff Visualization

The main purpose of Codeopticon is to enable a tutor to remotely monitor a group of learners while they are coding. If the tutor were monitoring only one learner, then this task would be trivial: display a live view of the learner's code, so that the tutor can see how it evolves as though they were sitting next to the learner in person. However, when the tutor needs to monitor several learners at once (design goal D2), they cannot keep up with everyone's code edits in real time.

To help the tutor follow everyone's edits, we have developed a novel real-time text diff visualization. *Text diffs* are a classic idea where the differences between two textual documents are compactly represented as blocks of character insertions and deletions. Code review and version control systems such as the GitHub Web interface allow users to visually compare multiple pieces of code by showing diffs, usually with insertions in green and deletions in red. However, to our knowledge, all existing diff tools are *static* – comparing two preset pieces of text, not text that is being actively edited while

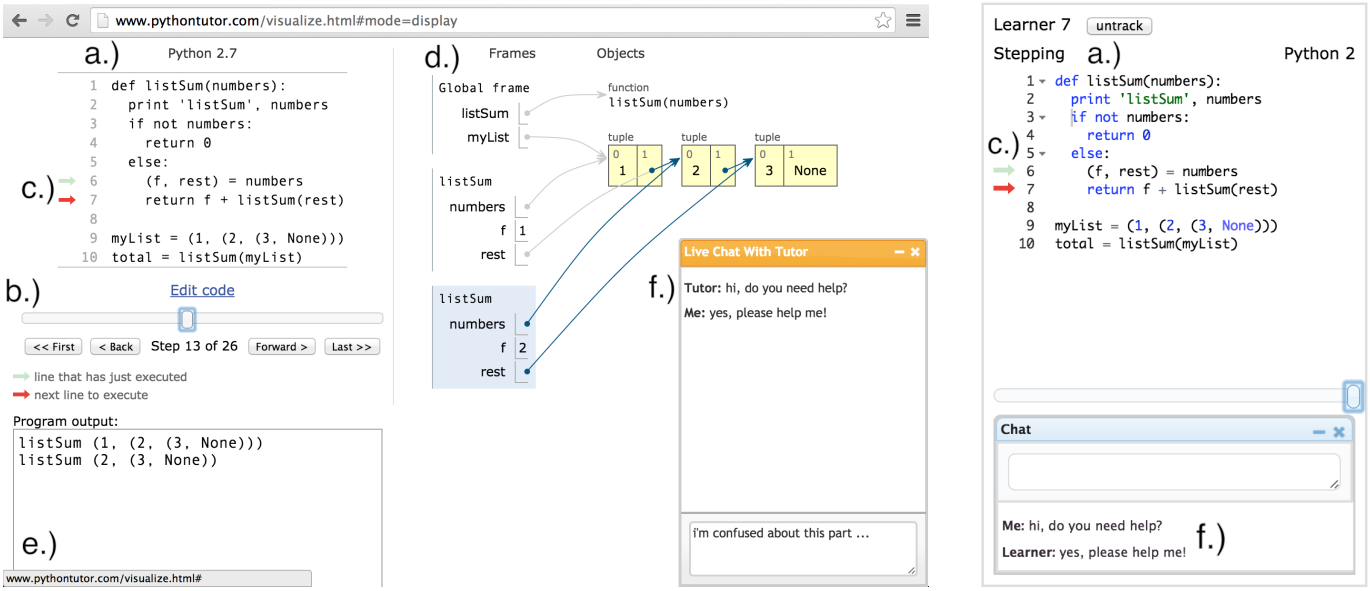

Figure 3. Each learner works in their own Online Python Tutor workspace (left). Codeopticon augments it with an embedded chat window (f.). The tutor sees each learner's actions in a tile within Codeopticon (right), with a real-time view of the a.) code editor, c.) currently executing line, and f.) chat.

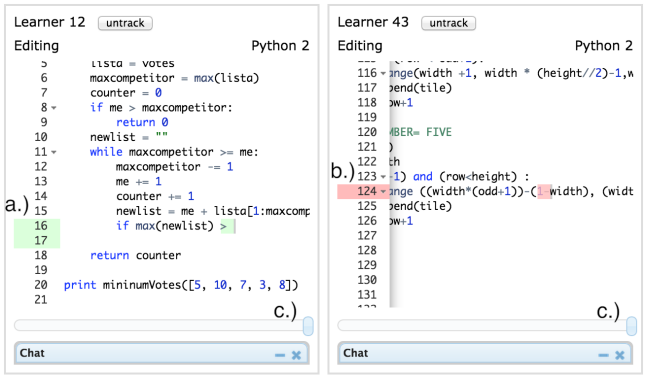

Figure 4. Within each tile, a.) code insertions are shown in green, and b.) deletions are shown in red (gutter highlights are useful here since the tutor needs to scroll to the right to see the actual diff). c.) A slider scrubs through each learner's full activity history. The chat box is minimized.

someone is watching. We created the following dynamic approach to visualizing text diffs within each tile (Figure 4):

- ***Debouncing Diffs*** – We compute diffs using Google's diff-match-patch library, which implements Myers' diff algorithm[1]. A naive approach would compute an insertion or deletion diff whenever the learner types or erases each individual character, respectively. To provide more meaningful diffs, we apply *debouncing* by computing a diff only after the learner stops typing for at least 1 second. In practice, this creates diffs consisting of entire tokens or phrases.
- ***Visualizing Insertions and Deletions*** – After debouncing, every diff shows up in real time within its tile, with insertions in green and deletions in red. For example, as a learner types out a function definition in Python, the tutor sees the following (debounced) insertions in green:

```
1  def
1  def listSum(
1  def listSum(num):
```

If the learner now decides to erase the `num` argument, the display alternates between showing the deleted text in red and eliminating it entirely, changing every 0.75 seconds:

```
1  def listSum(num):
1  def listSum():
1  def listSum(num):
1  def listSum():
```

It is important to show the state of the code both before and after the deletion, since if the tutor sees only the code after the deletion, they do not know what has been deleted.

- ***Gutter Highlights*** – Since diffs are sometimes tiny (a few characters) and can appear anywhere within the line, it can be hard to see them at a glance without first knowing which lines to look at. To focus the tutor's attention, the gutter to the left of each line of code turns green or red when text in that line has been inserted or deleted, respectively (Figure 4). When both insertions and deletions occur on the same line, the gutter shows red-and-green stripes.
- ***Auto-Scrolling*** – Each tile has space to show only around 20 lines of code. When an edit occurs, the tile automatically scrolls vertically to center around the changed line. Without auto-scrolling, the tutor cannot immediately see edits made to lines that are outside of the current viewport. Codeopticon is designed to show small pieces of pedagogical code that fit in a single file, not multi-file projects. Of the 2,697 total pieces of executed code in our user study, the median length was 37 lines (mean=64). Since each tile shows 20 lines and auto-scrolls, it is reasonably-sized for displaying typical learner-written code.

[1] code.google.com/p/google-diff-match-patch/

- ***History Slider*** – The tutor can see all past activity for a learner by dragging the slider at the bottom of their tile (Figure 4c), and the display will update to show past states. This navigation feature is analogous to a scrubber on a video player. Each learner's history includes not only their code diffs, but also their execution attempts, errors, and stepping through execution points in the visual debugger.

Taken together, these features enable a tutor to see how multiple learners edit their own independent pieces of code.

Dynamic Tile Reshuffling

Codeopticon comfortably fits 10 tiles on a laptop screen (e.g., Figure 1) and 15 tiles on a 30-inch monitor. This default setup works well as long as there are ≤ 15 learners. But an online coding workspace or MOOC can have dozens or hundreds of learners signed on at once (Figure 2), and even the TAs we observed had up to fifty students in each physical lab session. Thus, for Codeopticon to be useful in realistic settings, we need a more scalable way to manage tiles (design goal D1).

We iterated on several designs while testing on live data from the Online Python Tutor website, starting with the most naive approach of simply adding more rows of tiles to the bottom as new learners sign on. The downside of this first approach is that the tutor needs to scroll down to the bottom to see the newest learners, and then scroll back up to see older learners. Scrolling is disorienting since the tutor cannot rely on spatial memory to recall an absolute position for each learner.

Fragmentation was another problem we encountered. Since learners arrive and leave at unpredictable times and vary greatly in their levels of activity (design goal D3), some tiles would be bustling with activity, while others would be inactive or even empty. Since only a small percentage of learners are active at any given time, the tutor must scroll through dozens of stale tiles to find the few active ones.

To minimize scrolling and fragmentation, our next design grouped the most active learners together in the first few rows so that a tutor always sees the most active tiles without scrolling. It periodically sorted all tiles by decreasing levels of activity. However, this sorting approach created a new problem: tiles moved around in unpredictable ways, so the tutor would often lose track of learners they were tracking. We wanted Codeopticon to minimize unnecessary tile movement so that the tutor can make better use of their spatial memory. Figure 5 shows an overview of its final dynamic tile reshuffling algorithm, which we now describe in detail:

The *main table* is configured to fit comfortably on the tutor's monitor (15 tiles shown in Figure 5). It always displays the most active learners' tiles. If the tutor never scrolls, they will always be observing the 15 currently most active learners.

Each *overflow row* appears below the main table, numbered O1, O2, O3, O4, etc. Each row displays learners of decreasing levels of activity. For instance, learners in O1 are less active than those in the main table, learners in O2 are less active than those in O1, and O3 less than O2. Thus, when a tutor gradually scrolls downward, they will start observing learners in overflow rows, starting with the most active (O1).

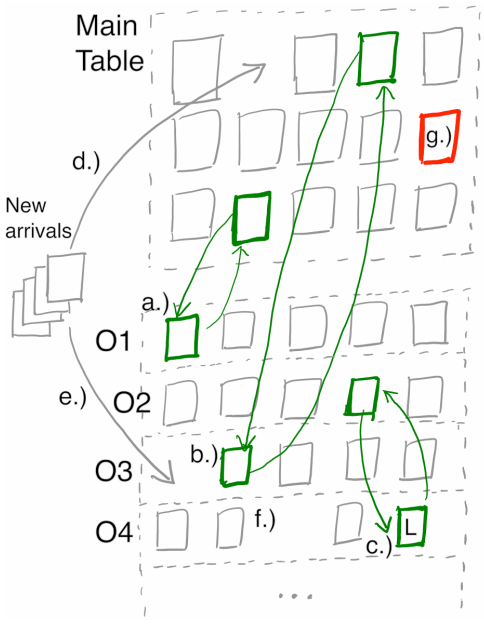

Figure 5. Codeopticon reshuffles tiles so that the most active learners are always in the main table, which the tutor can monitor without scrolling.

We need an activity metric to compare learners. The simplest is *events per second*, where an event is a code edit, execution, or debugger step. But we want to prioritize learners with more recent activity, so we use an exponentially-weighted average: When computing (weighted) events per second, each event receives a weight of $e^{-\lambda\Delta}$, where λ is how much to penalize older events (default of 0.1) and Δ is how many seconds ago this event occurred. If two learners have the same number of events, the one with more recent activity will be considered more active. Then periodically, every 10 seconds, the following algorithm runs to reshuffle tiles by relative activity levels:

- Figures 5a. and b. – If any learner in an overflow row is now more active than some learner in the main table (according to the exponentially-weighted average events metric), then swap their tiles. To minimize back-and-forth jitter and make the main table more stable, we give an additional *incumbent advantage* to learners who are already in the main table. To kick a tile out of the main table, a learner in an overflow row tile must be at least 1.3 times more active.

- Figure 5c. – If a learner cannot move to the main table, it moves up to the most active overflow row. Here a learner L in O4 moves up to O2. Note that L is also more active than everyone in O3, but our algorithm tries to move it up as far as possible, so it moves to O2. However, L cannot move to O1 or to the main table since it is not active enough.

- To minimize movement, no tiles within the main table ever swap places with one another, and neither do tiles within each overflow row; swaps occur only *across* overflow rows or between an overflow row and the main table. Thus, once a tile is in, say, the main table, it will not move unless kicked out by a more active tile from an overflow row or by a newly-arriving learner (see below).

Learners Arriving: So far we have described steady-state behavior, but what about when new learners sign on? Since most learners are low-activity, if they are added to the main table, they will quickly be demoted to an overflow row, which causes unnecessary jitter. Thus, each new learner gets a 15-second *probationary period*. If they have enough activity in the first 15 seconds (e.g., more than 4 events), then their tile moves to the main table (Figure 5d.); otherwise they get sent to the first overflow row where there is a free spot (Figure 5e.), or to a newly-created row if no more spots are free.

Learners Leaving: When a learner closes their Online Python Tutor browser window, their corresponding Codeopticon tile gets destroyed immediately. To minimize movement, all other tiles remain in the same place, so a hole is left in its wake (Figure 5f.). Holes serve as free spots for new learners to enter without needing to swap out someone else's tile, which further reduces movement and fragmentation.

Tile Locking: What if the tutor does not want a tile to move because they are closely monitoring that learner? They can click on any tile's border to lock it, which highlights it with a red border (Figure 5g. and Figure 1d.). Locked tiles are ignored by the reshuffling algorithm, so they will never move anywhere. Whenever a chat box is open, that tile auto-locks. Clicking on a tile again unlocks it and restores its default gray border, and clicking on the "untrack" button destroys it.

Grace Period: Unless a tutor locks all tiles they are observing, they can be glancing at a tile and then suddenly it gets swapped out without warning, making them lose track of the learner they were watching. To eliminate this jarring feeling, when a pair of tiles is scheduled to be swapped, both are temporarily highlighted in green for 5 seconds before the swap occurs (see Figures 5a., b., and c.). During this grace period, the tutor can click on a tile to lock it and cancel the swap.

USER STUDY OF NATURALISTIC ONLINE TUTORING

To see whether tutors can use Codeopticon to help learners in a naturalistic online setting, we conducted a user study where each subject spent 30 minutes tutoring learners on the Online Python Tutor website and then reflected on their experience.

Methodology

Since Codeopticon can be used in diverse settings ranging from a residential course to a MOOC to a paid online tutoring service, we considered several methodologies: A controlled lab study would provide the highest-fidelity data from in-person observations. However, this setup would not capture the scale and heterogeneity of a naturalistic online learner population that Codeopticon was designed to handle, with hundreds of learners signing on and off at unpredictable times and each exhibiting wildly varying activity levels. Deploying Codeopticon in a MOOC would capture that scale and heterogeneity, but such deployment is logistically difficult and does not allow us to directly observe the tutors working in-person.

We adopted a hybrid approach by bringing 8 subjects into our lab and having each spend 30 minutes anonymously tutoring learners who visited the Online Python Tutor website. That way, we could directly observe tutors at work while also capturing the diversity of a typical online learner population. Over 30,000 unique learners visit `pythontutor.com` each month to write and debug code, dozens are online at once, and hundreds arrive and leave per hour (Figure 2).

Subjects: We recruited 8 subjects (6 male, 2 female) from our university's Python-based introductory computer programming course: the instructor and 7 undergraduate lab TAs. Each subject received a $15 gift certificate for a 1-hour study.

Setup: Each subject used Codeopticon in Google Chrome on a 2014 Mac Mini desktop computer with a 30-inch monitor at 2560x1600 resolution, with 15 tiles in the main table (3x5).

Task: We gave each subject a 10-minute tutorial on Codeopticon, since none had seen it before. Then we instructed: *"Spend the next 30 minutes helping as many learners as you can in whatever way you feel most comfortable."* We encouraged them to think aloud. We did not know who would be visiting the Online Python Tutor website at that time, or what code they would be writing. Although learners were informed that their actions were being monitored for research, they did not know when (or if) a tutor would start a chat with them. Thus, this interaction resembles a visitor on a shopping website seeing a pop-up chat box from a customer service representative offering live help. To focus our study on tutor-initiated actions, learners cannot summon a tutor for help.

Post-Task Interview: The final 15 minutes of each session was a semi-structured interview where we asked the subject to discuss the rationale behind some of the specific interactions that we observed (e.g., "why did you keep scrolling back and forth between the overflow rows and main table?"). We also asked three open-ended questions to seed further discussion: 1.) How is Codeopticon better than your in-person Python tutoring experiences? 2.) How is it worse than in-person tutoring? 3.) How would you adapt Codeopticon to deploy in a residential or online course that you teach in the future?

Quantitative Results

Since the goal of our study was to observe how tutors used Codeopticon in a naturalistic setting, we did not run controlled trials to compare different user interface conditions or give monetary incentives for, say, helping more learners. Thus, these quantitative results represent an informal, holistic view of Codeopticon usage, not claims about the efficacy of specific features. Table 1 summarizes the sessions.

The 8 subjects are sorted by years of self-reported Python experience, with TA1 and TA2 having only .25 years and the instructor (INS) with 12 years. During their 30-minute session, each subject saw a mean of 226 different learners writing code on the Online Python Tutor website, with 52 online at any given moment. Since each subject participated in the study at a different time, they likely saw a completely different set of learners. We had no control over who was online in each session, or how receptive they were to unsolicited help.

The numbers in the "# Chat Conversations Initiated" columns represent each subject's intent to help specific learners. Each subject initiated, on average, 12 conversations. As we observed from think-aloud and post-task interviews, all subjects

Name	Years of Python Experience	# Learners Seen		# Chat Conversations Initiated			Confirm Helped	# Chat Msgs.	
		Total	Avg. Online	All	w/ Generic Hello	w/ Proactive Help		Sent	Received
TA1	.25	241	57	7	7	0	0	7	1
TA2	.25	220	50	37	37	0	5	89	50
TA3	.5	201	50	9	1	8	2	17	6
TA4	1	258	62	7	0	7	1	18	24
TA5	1	181	46	6	0	6	2	17	9
TA6	2	200	41	10	3	7	3	42	18
TA7	3	223	51	10	0	10	1	14	1
INS	12	283	58	8	0	8	5	31	37
Average	2.5	226	52	12	6	6	2.4	29	18

Table 1. Results from a user study where eight subjects each spent 30 minutes using Codeopticon to tutor learners on the Online Python Tutor website.

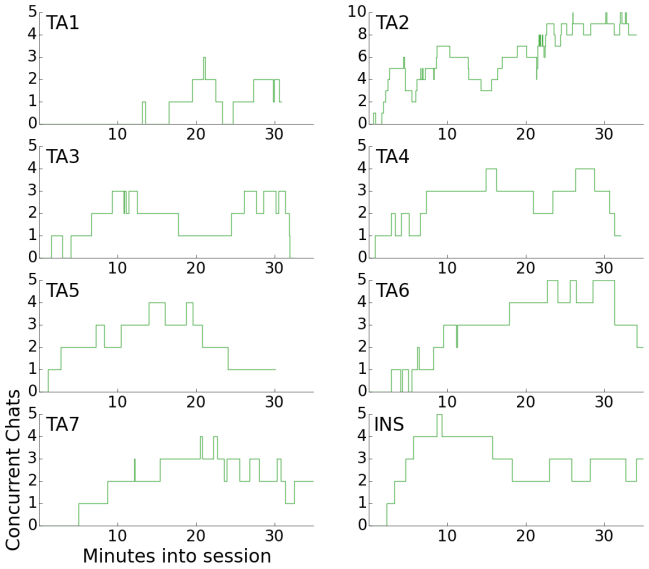

Figure 6. Number of concurrent chat conversations during each session.

(except for TA2) carefully studied a learner's actions in their respective tile and only initiated a chat when they were sure that they knew the problem the learner was facing. They were reluctant to disturb learners unless they felt like they could genuinely help. TA2 was the notable exception, aggressively initiating chats with most learners in the main table using the same generic greeting: *"Would you like some help?"*

The least experienced subjects (TA1 and TA2) initiated chats with generic hello messages. Others initiated most chats with a proactive help message. For instance, INS noticed that a learner was getting a hard-to-debug syntax error since a string value was incorrectly quoted using curly quotes ("`foo`") rather than plain quotes ("`foo`"), so they started a chat with: *"Hi! If you pasted this [code] in, you are using 'curly quotes' or 'smart quotes.' Try deleting those quotes and try again."* TA6 started off with generic hello messages but then switched to proactive help, mentioning during the post-task interview that, "I'm pretty sure learners thought I was a bot when I greeted them with a generic hello, so that's why I wanted to craft specific help messages to show that I was a human."

Figure 6 shows the number of concurrent chat conversations throughout each subject's session. Each conversation ends whenever either the subject or the learner closes the chat window, or when the learner signs off. Several subjects went a few minutes over their 30-minute time limit since they were busy wrapping up conversations. Most subjects had two to five chat windows open at once, except for TA2, who took the aggressive approach of messaging many learners at once.

These numbers show that first-time Codeopticon users are able to sustain multiple concurrent chats and help learners in a naturalistic online setting, despite the learners not expecting to receive help. Online learners might be reluctant to respond to unsolicited pop-up chat messages. They might also not know English, since IP addresses indicated that they came from 54 countries. If Codeopticon were deployed in a formal course where TAs and students knew about this system beforehand, we expect help confirmation rates to be higher.

Indicators of Learning

The "Confirm Helped" column of Table 1 shows that 19 total learners confirmed that they got helped, usually by fixing a bug and then replying with a thank-you note. One enthusiastic learner ended their chat with: *"thank you so much. We are pretty understaffed on TA's for our class size so I really like that you have this [service] available."* Those who received help seemed engaged: 4 of 19 asked follow-up questions, and two others even asked the tutor how they could request their help later. In addition, several more appeared to take the tutor's proactive help by editing their code, but we did not count those cases in Table 1 since they did not reply to confirm.

We read all 19 confirmed help instances in the chat logs and coded for *indicators* of learning. However, we cannot make any claims about true learning since we did not run any formal assessments. And since learners were anonymous, we did not interview them to get firsthand accounts of their impressions.

All learning instances were at the lowest two levels of Bloom's taxonomy [4]: remembering and understanding. This finding was unsurprising because we did not expect these brief, low-bandwidth chat sessions to contain substantive learning at the higher levels of Bloom's taxonomy: applying, analyzing, evaluating, and creating. In sum, Codeopticon was good for tutors to help learners debug their code and get momentarily unstuck, but not for facilitating deep pedagogy.

9 of 19 help instances involved helping the learner *remember* a piece of Python syntax. Tutors were effective at translating Python's default unhelpful error messages (e.g., "SyntaxError: invalid syntax") into more meaningful ones, especially when the true error was on a different line than indicated. One tutor helped a learner clarify brackets and parenthesis usage: "Hi there! It looks like you're trying to use the range function, which doesn't use square brackets. Try range(...)"

10 of 19 help instances involved *understanding* Python runtime semantics. Example concepts that tutors successfully conveyed include global versus local variable scoping, type mismatch errors, off-by-one error in iteration, mutable versus immutable sequences (i.e., list versus tuple in Python), using list comprehensions instead of a for-loop, and basic recursion.

Tutors' Qualitative Impressions of Codeopticon Interface

From our observations, think aloud, and post-task interviews, all subjects appeared to find Codeopticon intuitive to use. It was easy for them to understand the metaphor of each tile being a virtual window into a learner's workspace, and everyone was already familiar with the idea of embedded chat.

After an initial period of lurking, subjects spent most of their session focused on chatting with learners (Figure 6). As expected, subjects spent most time on the main table (Figure 5), but occasionally they scrolled down to the overflow rows during pauses in the current chat conversations to see if any learners there had triggered error messages.

Several subjects expressed a strong urge to directly edit the learner's code within each tile in addition to making suggestions via chat. However, during post-task discussion, they acknowledged that it would be disconcerting for a learner to see someone else edit their code, akin to an in-person tutor rudely taking the keyboard from them and saying "no, let me show you how to do it!" Instead, subjects proposed either being able to highlight a learner's code to direct their attention, or turning edits into suggestions that a learner can either accept or reject (like the Track Changes mode in Microsoft Word).

Several subjects wanted to enlarge a learner's tile when they were chatting, so that they could see more lines of code and not get distracted by other tiles. They suggested a zoomed-in mode where only tiles with active chats are shown. INS mentioned that such a focused view would "ameliorate my guilt a bit [at not being able to help everyone I saw]. Out of sight, out of mind." However, other subjects liked being able to browse around all of the tiles during lulls in chat conversations.

Attention: Although we did not formally measure attention, it appeared like each subject could sustain at most three real tutoring interactions at once. (They sometimes had more chat windows open but were waiting for learner responses.) This is unsurprising, since tutoring requires high concentration, and Codeopticon does not somehow enhance this mental capacity. Rather, Codeopticon's strength is in directing the tutor's attention to the learners who might need help at the moment.

Fatigue: We did not run a formal stress or endurance test, but nobody appeared fatigued after 30 minutes of using Codeopticon at their own natural pace. In fact, Figure 6 shows that all subjects except for TA1 and TA5 voluntarily went over the 30-minute limit to continue chatting with learners.

Authenticity and Immersion: Even though subjects knew they were chatting with anonymous strangers for a user study, they appeared to be immersed in the task as though they were tutoring in real life, with their own personal styles showing through in conversations. They showed pride upon receiving thank-you messages from learners, frustration when someone was not following their suggestions, and hesitation when unsure of whether they should interrupt a learner at work. For instance, TA2 adopted a highly proactive style of chatting with lots of learners; when asked about their rationale for doing so, they replied that this was exactly the sort of energetic approach they took in lab. In contrast, TA1 started no chats for almost 15 minutes, and then only said a few generic hellos; that reflected their style of waiting for learners to come to them in lab rather than risk disturbing anyone in the midst of working. INS (the course instructor) was used to lecturing, so they turned some conversations into impromptu mini-lectures on topics such as list versus tuple data structures in Python.

Advantages of Codeopticon Over In-Person Tutoring

During the post-task interviews, subjects pointed to several advantages of Codeopticon over in-person tutoring. Most importantly, it allows the tutor to monitor and reach out to shy learners. In contrast, it is awkward to be peering over someone's shoulder in a computer lab. Several subjects mentioned how they wish they could use this in their own lab sessions with everyone identified by their real names, since they suspected that some of their students would benefit from their help but were reluctant to ask in person. But other subjects felt that remaining anonymous would actually encourage more questions, since "it can be terrifying to ask a super-basic question to an experienced programmer."

Codeopticon also enables a tutor to control their own pace, chatting with as many or as few learners as they wish. A single overly-demanding learner cannot easily monopolize the tutor's time, as they sometimes do in a computer lab. Similarly, it also enables learners to keep working while waiting for the tutor to respond to chats; it can be hard for some learners to code while a tutor is sitting beside them in person.

Subjects also liked using the history slider (Figure 4c) to see how a learner's prior actions led to the current state, so that they know what alternatives the learner attempted. Doing this in-person is impossible or at least disruptive, since it requires the learner to stop coding and undo edits in their code editor.

Subjects mentioned how they could use Codeopticon to schedule virtual lab hours, and how they could privately search the Web for extra help without fear of embarrassment. INS said, "one of my main problems with a big class is that I can't work one-on-one with students, but with this [tool], I can sit in the comfort of my own office or even at home and work one-on-one with students who are in lab."

Finally, subjects felt that being able to see dozens of learners at once (essentially an entire classroom or lab section) was very powerful, not only for helping individual learners, but

also for seeing whether everyone was on track and making steady progress toward their given assignments.

Limitations of Codeopticon

The main limitation subjects mentioned was that text-based chat was far lower bandwidth than face-to-face conversation. It was hard to provide detailed conceptual explanations, especially those that required sketching diagrams. One improvement would be to connect tutors with learners on Google Hangouts (like Talkabout [15]) or a shared drawing canvas.

Subjects also cited a tradeoff between quantity and quality of tutoring: Codeopticon allows a tutor to help more learners in a somewhat superficial way via text chats, but in-person tutoring allows deeper focus on fewer learners. INS said that with Codeopticon it was hard to focus on just one learner, since "my good nature wanted to come out. [Upon seeing someone else's activity ...] ohhh I want to help you too!"

Another limitation is that the tile reshuffling algorithm biases the tutor toward learners with greater edit activity. Perhaps some learners are stuck and staring blankly at their screens; but Codeopticon cannot tell whether those idle learners actually need help or are simply browsing the Web. One way to ameliorate is to enable a learner to summon the tutor for help, which will move their tile up to the main table. Also, the algorithm could prioritize learners who are excessively scrolling, focusing and unfocusing windows, or whose activity level suddenly drops, which could indicate frustration.

CONCLUSION

Codeopticon is a prototype system that enables a single tutor to monitor and chat with dozens of learners who are coding in an online workspace. A study demonstrated that first-time users can successfully tutor a variety of anonymous learners in a naturalistic online setting. If this kind of system were integrated into a MOOC or in-person course, then TAs could schedule virtual office hours to provide timely, targeted, and proactive help at a greater scale than is currently feasible.

ACKNOWLEDGMENTS

Thanks to Parmit Chilana, Logan Gittelson, Mitchell Gordon, Juho Kim, and Chris Parnin for their feedback. This work was supported in part by the National Science Foundation under grant NSF CRII IIS-1463864.

REFERENCES

1. 2013. How big is UC Berkeley's biggest class? *The Daily Californian* (Sept. 2013).
2. 2015. The XY Problem. (April 2015). `xyproblem.info`
3. Susan A. Ambrose, Michael W. Bridges, Michele DiPietro, Marsha C. Lovett, and Marie K. Norman. 2010. *How Learning Works: Seven Research-Based Principles for Smart Teaching* (1 ed.). Jossey-Bass.
4. Lorin W. Anderson and David R. Krathwohl (Eds.). 2000. *A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives.* Allyn & Bacon.
5. Derrick Coetzee, Armando Fox, Marti A. Hearst, and Björn Hartmann. 2014. Chatrooms in MOOCs: All Talk and No Action *(L@S '14)*. 127–136.
6. Derrick Coetzee, Seongtaek Lim, Armando Fox, Björn Hartmann, and Marti Hearst. Structuring Interactions for Large-Scale Synchronous Peer Learning *(CSCW '15)*.
7. Paul Dourish and Victoria Bellotti. 1992. Awareness and Coordination in Shared Workspaces *(CSCW '92)*.
8. Elena L. Glassman, Jeremy Scott, Rishabh Singh, Philip J. Guo, and Robert C. Miller. 2015. OverCode: Visualizing Variation in Student Solutions to Programming Problems at Scale. *TOCHI* 22, 2 (2015).
9. Max Goldman, Greg Little, and Robert C. Miller. Real-time Collaborative Coding in a Web IDE *(UIST '11)*.
10. Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-based Program Visualization for CS Education *(SIGCSE '13)*. ACM, 579–584.
11. Mark Guzdial. 2014. Limitations of MOOCs for Computing Education - Addressing Our Needs: MOOCs and Technology to Advance Learning and Learning Research (Ubiquity Symposium). *Ubiquity* (2014).
12. R. A. Hines and C. E. Pearl. 2004. Increasing interaction in web-based instruction: Using synchronous chats and asynchronous discussions. *Rural Special Education Quarterly* 23, 2 (March 2004).
13. Juho Kim, Elena L. Glassman, Andrés Monroy-Hernández, and Meredith Ringel Morris. 2015. RIMES: Embedding Interactive Multimedia Exercises in Lecture Videos *(CHI '15)*.
14. René F. Kizilcec, Chris Piech, and Emily Schneider. 2013. Deconstructing Disengagement: Analyzing Learner Subpopulations in Massive Open Online Courses *(LAK '13)*. 170–179.
15. Chinmay Kulkarni, Julia Cambre, Yasmine Kotturi, Michael S. Bernstein, and Scott R. Klemmer. 2015. Talkabout: Making distance matter with small groups in massive classes *(CSCW '15)*.
16. Thomas D. LaToza, Ben Towne, Christian M. Adriano, and André van der Hoek. Microtask Programming: Building Software with a Crowd *(UIST '14)*.
17. Jeffrey Rzeszotarski and Aniket Kittur. 2012. CrowdScape: Interactively Visualizing User Behavior and Output *(UIST '12)*. ACM, 55–62.
18. N. B. Sarter and D. D. Woods. 1991. Situation awareness: A critical but ill-defined phenomenon. *Int. Jour. Aviation Psychology* 1, 1 (1991), 45–57.
19. Juha Sorva. 2012. *Visual Program Simulation in Introductory Programming Education*. Ph.D. Dissertation. Aalto University.
20. Sarah Vieweg, Amanda L. Hughes, Kate Starbird, and Leysia Palen. 2010. Microblogging During Two Natural Hazards Events: What Twitter May Contribute to Situational Awareness *(CHI '10)*. 1079–1088.
21. Andreas Zeller. 2005. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers.