# Toward a Domain-Specific Visual Discussion Forum for Learning Computer Programming: An Empirical Study of a Popular MOOC Forum

Joyce Zhu, Jeremy Warner, Mitchell Gordon, Jeffery White, Renan Zanelatto, Philip J. Guo
Department of Computer Science
University of Rochester
Rochester, NY 14627
{jzhu29,jwarn10,mgord12}@u.rochester.edu, {jwhite37,rzanelat}@ur.rochester.edu, pg@cs.rochester.edu

***Abstract*—Online discussion forums are one of the most ubiquitous kinds of resources for people who are learning computer programming. However, their user interface – a hierarchy of textual threads – has not changed much in the past four decades. We argue that generic forum interfaces are cumbersome for learning programming and that there is a need for a domain-specific visual discussion forum for programming. We support this argument with an empirical study of all 5,377 forum threads in *Introduction to Computer Science and Programming Using Python*, a popular edX MOOC. Specifically, we investigated how forum participants were hampered by its text-based format. Most notably, people often wanted to discuss questions about dynamic execution state – what happens "under the hood" as the computer runs code. We propose that a better forum for learning programming should be visual and domain-specific, integrating automatically-generated visualizations of execution state and enabling inline annotations of source code and output.**

***Keywords*—discussion forums, MOOC, CS education**

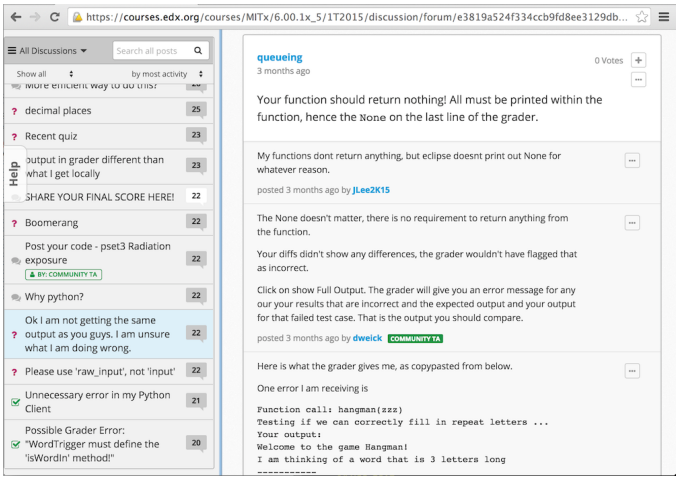


Fig. 1. A screenshot from the forum of a MOOC on computer programming [11], which looks like a typical online discussion forum consisting of topics, threads, replies, and mechanisms for voting, searching, and filtering.

## I. Introduction

Online discussion forums are one of the most ubiquitous kinds of resources for people who are learning computer programming. Both novices and experts search the Web extensively while they are coding in order to learn the nuances behind the programming languages, libraries, and frameworks they are using [1], [2]. Many programming-related Web searches lead to some sort of discussion forum: The Q&A forum StackOverflow is one of the most popular [3], [4], but thousands of niche forums exist for every conceivable piece of programming technology. In addition, online documentation pages, technical blog posts, and open-source code repository websites often embed discussion forums at the bottom of each webpage to allow people to discuss that page's contents.

Discussion forums also play a central role in online computing education initiatives such as MOOCs (Massive Open Online Courses) [5], [6], Codecademy [7], and Khan Academy CS [8]. Unlike those in traditional classrooms, instructors in large-scale online settings cannot give high-fidelity, personalized, real-time help to the tens of thousands of learners who are visiting educational websites. The asynchronous nature of discussion forums allows learners to post questions and help one another at their own convenience. Instructors and teaching assistants also contribute to and moderate forums. Aside from being a resource for technical answers, forums are crucial for fostering a sense of camaraderie and social bonding for online learners who never meet their classmates face-to-face [5].

Despite the widespread importance and usage of discussion forums, their user interface has not changed much in the past four decades since early incarnations such as Usenet newsgroups [9] and The WELL [10]. Figure 1 shows a forum from an introductory computer programming MOOC on the edX platform [11]. This forum is comprised of a tree of text-based threads; other kinds of programming forums look almost identical. Some popular sites such as StackOverflow have incorporated features such as voting, reputation metrics, category tags, moderation, searching, sorting, filtering, rich-text formatting, and syntax highlighting for code. But at their core, most forums are simply a generic tree of textual discussions. The exact same piece of forum software could be used for a computer programming class as for a fan discussion page about the latest celebrity gossip.

In this paper, we argue that this sort of generic discussion forum is cumbersome for discussing common computer programming topics and that there is a need for a domain-specific visual discussion forum for learning programming. We support this argument by presenting an empirical study of all 5,377 forum threads in the Spring 2015 offering of *Introduction*

| Finding | Design Recommendation |
|---|---|
| The majority of discussion forum threads in this course (∼60%) mentioned execution state, code snippets, output/errors, or autograder complaints. | A forum optimized for learning programming must support inline anchored discussions around these topics, eliminating the need to copy-and-paste into disconnected threads. |
| Generic forums have no built-in way of visualizing execution state, so posters must resort to indirect textual explanations or manually-drawn diagrams of state. | A forum should integrate with a program visualization tool that automatically renders diagrams of code execution state, eliminating the need for manually-drawn diagrams. |
| Generic forums are not designed for holding rich conversations around source code; they treat code simply as plain text with some optional formatting. | A forum should treat code not as blocks of plain text, but rather as first-class objects that can be annotated, versioned, and linked to code posted by other students. |
| Generic forums are not linked with a code execution engine, so they cannot effectively capture discussions and experimentation around program output/errors. | A forum should tightly integrate with an IDE (Integrated Development Environment) so that students can execute code, see output/errors, and discuss them within the IDE. |
| The autograder is a central component of programming courses, but how it works is often unclear to students. Thus, some discussions involve complaints about it. | A program visualization tool can visualize how the autograder works; integrating the forum with these visualizations allows students to more easily discuss grading issues. |

TABLE I. SUMMARY OF FINDINGS AND ACCOMPANYING DESIGN RECOMMENDATIONS FROM OUR STUDY OF 5,377 DISCUSSION FORUM POSTS IN A POPULAR COMPUTER PROGRAMMING MOOC.

*to Computer Science and Programming Using Python* [11], a popular MOOC released by MIT on the edX platform.

Specifically, we investigated how forum participants were hampered by the limitations of its text-based format. Most notably, people often wanted to discuss questions about *dynamic execution state* – what happens "under the hood" as the computer runs a piece of code. This observation corroborates the fact that one of the fundamental challenges of learning programming is developing a robust mental model of dynamic execution state [12], [13]; without good mental models, novices are susceptible to hundreds of well-documented misconceptions about how their code works [14] and cannot write reliable programs of any significant size. But it is hard to talk about execution state in an ordinary text-based forum since run-time concepts such as stack frames, variables, pointers, objects, and data structure shapes are invisible. Students must now copy-and-paste snippets of code and text output from the terminal into their posts, and then indirectly talk about run-time semantics. There is no easy way to visualize this state.

Table I summarizes all of our study's findings and accompanying design recommendations. In sum, we propose that a better forum for learning programming should be visual and domain-specific in nature, tightly integrating automatically-generated visualizations of execution state [15] and enabling inline annotations of both source code and program output.

Discussion forums are one of the longest-enduring forms of online knowledge sharing, but a purely text-based format ignores the rich nuances of each domain and hinders their potential as an educational tool. There is no one-size-fits-all solution, though; intuitively, the ideal forum for learning world history should look very different from the ideal one for learning computer programming. Our goal in this paper is to uncover some of the limitations of using generic forums for computing education in particular.

Our study is the first step toward informing the design of the next generation of forums for programming, which can benefit a broad audience due to the growing importance of computational thinking across many fields of work [16]. As more people of all backgrounds learn programming from online resources, it is important to provide them with the best possible medium for seeking help and fostering discussions, especially about technical topics that are cumbersome to discuss using generic forum interfaces.

This paper's contributions are:

- An empirical study of all 5,377 discussion forum posts in a popular computer programming MOOC, which shows how a generic text-based forum format is cumbersome for discussing four topics that were mentioned in many threads: execution state, code snippets, program output/errors, and autograder complaints.
- A proposal for the design of a new kind of domain-specific visual discussion forum for learning programming, informed by the findings of our study. Our forum design integrates inline anchored discussions and program visualizations into a Web-based IDE.

## II. BACKGROUND AND RELATED WORK

Online discussion forums have been a staple of Internet culture for almost four decades, starting with Usenet newsgroups and bulletin board systems (BBS) in the 1970s [9] and The WELL [10] in the 1980s. Due to limited computational power and network bandwidth at the time, these early forums were all text-based, with conversations grouped into hierarchies (trees) of threads. When Web forums started in the 1990s, they simply replicated this text-based threaded format that has now become ubiquitous. Numerous forums now exist for topics ranging from parenting [17] to mathematics [18]. In the 2000s, question-and-answer (Q&A) sites such as Yahoo! Answers [19], [20], StackOverflow [3], [4], the StackExchange network [21], [22], and Quora [23] grew popular. These sites share the same format as traditional forums, except that each thread starts with a question, followed by a series of answers that users can vote on so that the best one rises to the top.

Although the majority of forums are purely text-based, several kinds of niche forums have incorporated domain-specific interface features. For instance:

- Image-based forums (called imageboards) make it easy to post images alongside text. One of the most popular, 4chan, combines a focus on images with an ephemeral format where posts are usually deleted within 4 minutes as incoming posts replace them [24].
- StackOverflow [3], [4] and other forums for computer programming enable posters to write code in indented blocks with syntax highlighting to improve readability.

- Mathematics forums such as MathOverflow [18] enable posters to write math formulas in LaTeX.

One forum interface variant used in education is called *anchored discussions* [25], where each piece of course content (e.g., lecture note, video, assignment) is tightly connected to its own mini-forum. Guzdial and Turns created the CaMILE system and found that anchoring improved on-topic discussions and led to better learning in their classes [26]. Zyto et al. took this idea further with NB [27], a Web-based system that allows students to annotate and hold discussions directly in the margins of PDF documents. NB enables students to ask and answer questions in real-time while they are in the flow of reading online lecture notes or digital textbooks. Many modern online learning systems such as Khan Academy and MOOCs feature a combination of anchored and traditional forums.

Forums are the primary way in which students communicate with each other and with instructors in MOOCs [5]. Researchers have studied aspects of forums such as collaborative learning [28], reputation systems [6], power-user behavior [5], and read-only (lurking) behavior [29]. However, prior studies of MOOC forums have not focused on any particular domain of learning, but rather on general student behavior irrespective of subject matter. The study in this paper focuses on computer programming MOOCs, with an eye toward how to improve the forum's user interface to support discussions about programming-related topics.

Researchers in computer-mediated communication have studied the myriad ways in which people interact with one another on forums across diverse domains [19], [20], [28], [17], [3], [4], [21], [23], [18], [22], [5], [6]. Although many such studies take the forum's user interface as a given, several have suggested design improvements. For instance, researchers who study software bug tracking systems [30], [31], [32] and product support forums [33] have suggested improvements to these kinds of interfaces to improve the workflows of software developers and support specialists, respectively.

Our study focuses on how students use a text-based threaded forum to discuss common elements within a computer programming course and then proposes ways in which the forum's user interface could be improved to better accommodate these discussions. Our study is unique in that it turns a critical eye on discussion forum interfaces in the domain of online learning at scale, which prior work has not investigated, and suggests improvements for computing education in particular.

## III. Methodology

We analyzed discussion forum posts from the Spring 2015 offering of *MITx 6.00.1x: Introduction to Computer Science and Programming Using Python* [11], a free MOOC (Massive Open Online Course) released by MIT on the edX platform. We chose to study this course because it is an introductory programming MOOC from one of the major providers (edX), has been offered four times before, and is based on MIT's popular introductory programming course that is taken by both CS majors and non-majors. This course has no prerequisites and is targeted at absolute beginners, although it aims to be just as rigorous as its on-campus MIT counterpart [34].

This course ran for nine weeks, from January 7 to March 11, 2015. Each week, the staff released a new set of lecture videos and homework assignments that involved programming in the Python language. There was also a midterm and final exam. The discussion forum (Figure 1) was the officially-sanctioned way for students to communicate with one another and with the course staff. In total, 4,267 people posted messages to 5,377 threads. The forum was very active, with an average of 84 new threads being created every day, and each thread's initial post receiving an average of 1.8 replies.

Most people posting to the forum were normal students, but there was also one instructor and 5 assistants called *Community TAs*. A Community TA is a current student who has established a good reputation for being helpful, responsible, and respectful on the forum, so the instructor has given them moderation privileges. Unlike normal students, the instructor and Community TAs can edit and delete anyone's posts.

Aside from being a standalone section of the course website, the forum is also embedded within all other components of the course. For instance, when a student is watching a particular lecture video, they can ask questions about it at the bottom of the page, and those will automatically be posted to the forum and tagged with the proper context (e.g., "Week 2: Lecture 4"). This is an example of *anchored discussions* [25], [26], which brings discussions closer to the course content.

### A. Manually Labeling Forum Posts

We scraped data from the edX website and wrote scripts to automatically identify features such as poster identities, thread lengths, and screenshots. However, it was hard to automatically categorize the actual content of threads, so we relied on manual labeling from six experienced Python programmers.

First, two researchers – the first and last author – separately read a random sample of 200 threads and performed an open card sort to identify the most salient topics that were discussed in those threads, especially those that frequently led to *frustrations with the forum's user interface*. They converged on four topics that were present in many of those threads:

- **Execution state**: Students often discussed dynamic properties of run-time state by describing what they think happens at each step of execution. For instance, one post tried to explain some exception handling code: *"The else clause will only run if NO error occurs. In this instance the divide by 0 error occurs, so the else does not execute. The finally clause executes, then the divide by 0 is thrown to the system handler."* Confusions stemmed from students' inability to visualize what their words were referring to.
- **Code snippets**: Students wrote or copied-and-pasted snippets of code into posts. Some knew how to use proper markup to have code appear in formatted blocks, but others did not, so their code ended up looking badly formatted. This led to subtle misunderstandings since indentation is significant in Python.
- **Output/errors**: Students copied-and-pasted the textual output or errors from code execution into their posts. Again, improper text formatting was a cause of many frustrations, as was the sheer amount of output that some programs dumped to the terminal.

| | a.) Total | | b.) Not General/Week1 | | c.) Not General/Week1 and has replies | |
|---|---|---|---|---|---|---|
| # threads | 5,377 | | 4,421 | | 3,619 | |
| # threads with some topic | 3,072 | (57%) | 2,719 | (62%) | 2,599 | **(72%)** |
| # threads with **Execution state** | 1,452 | (27%) | 1,253 | (28%) | 1,220 | **(34%)** |
| # threads with **Code snippets** | 1,809 | (34%) | 1,589 | (36%) | 1,527 | **(42%)** |
| # threads with **Output/errors** | 839 | (16%) | 772 | (17%) | 759 | **(21%)** |
| # threads with **Autograder complaints** | 870 | (16%) | 799 | (18%) | 763 | **(21%)** |

TABLE II. THE NUMBER (AND PERCENT) OF THREADS IN THE DISCUSSION FORUM OF THE EDX MOOC *Introduction to Computer Science and Programming Using Python* THAT MENTIONED EACH OF OUR FOUR MANUALLY-LABELED TOPICS.

- **Autograder complaints**: Finally, students complained about the *autograder* – the automatic grading software that checks the correctness of every programming assignment. The autograder works by running the student's Python code on a collection of test inputs and comparing the outputs to instructor-created answers. Complaints often stemmed from the autograder interface being too opaque, simply telling the user whether each part was right or wrong, but not how or why.

After finalizing the topics, we recruited the remaining authors (six total) to manually label all 5,377 forum threads according to which of the above topics were present in each one. We split threads into six groups so that each researcher labeled 896 of them. All six were experienced Python programmers.

To determine inter-rater reliability, all six researchers labeled the same random sample of 50 threads (1% of total threads). The Fleiss' kappa scores amongst six raters were 0.40 for identifying a thread as containing execution state, 0.76 for code snippets, 0.57 for output/errors, and 0.62 for autograder complaints. 1.0 means perfect agreement, so these scores indicate moderate agreement. Two factors lowered our scores: having more raters tends to lower the scores, and there was subjectivity involved in picking out topics (especially mentions of execution state) embedded within blocks of text.

The rest of this paper describes our findings and design recommendations, which are summarized in Table I.

## IV. QUANTITATIVE FINDINGS: PREVALENCE OF TOPICS

Before describing specific examples of user frustrations with the forum's interface, we first present numbers to show how prevalent our four labeled topics were throughout the forum. Establishing prevalence is important because if these topics comprise only a tiny fraction of forum posts, then it is not worth trying to redesign future computing education discussion forums to accommodate them.

Table II shows that the majority of threads contained at least one of the four topics. The first column – "a.) Total" – considers all 5,377 threads in the course, where 57% were labeled with at least one topic. Many threads contained multiple topics. The three most common co-occurrences were: 70% of execution state threads also showed code snippets (thus conveying both static and dynamic properties of code), 44% of autograder complaint threads also showed output/errors, and 22% of execution state threads also showed output/errors.

To see how prevalent these topics were in the most active and relevant threads, we filtered using two criteria: First, many threads in Week 1 involved course logistics and software setup problems since there was not a programming assignment due yet; thus, that first week was not representative of the rest of the course. Also, threads posted to a catch-all "General" area of the forum were side discussions that had little bearing on the core course material. We filtered out those posts to create the second column in Table II – "b.) Not General/Week1" – where 62% had at least one labeled topic. Next, we saw that threads with no replies were often badly-worded, off-topic, or otherwise incomprehensible. Regardless of cause, students and teaching staff did not try to engage with those threads. We filtered those out to create the third column in Table II – "c.) Not General/Week1 and has replies." These are likely to be threads that covered core course material *and* had engagement. 72% of these threads had some labeled topic, and over one-third showed either execution state or code snippets.

| | | |
|---|---|---|
| # attached images | 257 | |
| # images with some topic | 175 | (68%) |
| # images with **Execution state** | 12 | (5%)† |
| # images with **Code snippets** | 54 | (21%) |
| # images with **Output/errors** | 69 | (27%) |
| # images with **Autograder complaints** | 40 | (15%) |

TABLE III. NUMBER OF IMAGES ATTACHED TO FORUM THREADS THAT DISPLAYED EACH TOPIC. († VERY FEW DISPLAYED EXECUTION STATE SINCE THE FORUM HAD NO BUILT-IN WAY OF VISUALIZING THIS STATE.)

**Image attachments**: In addition to mentioning these topics in the text itself, people sometimes attached images to posts to illustrate them. 5% of total threads included some image attachment, which renders inline alongside the text.

Table III shows that out of 257 attached images, 68% illustrated one of our four topics. Many attachments were screenshots taken of either the edX website or of an external piece of software. Only 5% showed execution state; this low proportion was likely due to the forum having no built-in means for visualizing such state, so posters had to turn to external tools. For instance, in Figure 2, the poster attached a screenshot from a Python program visualization tool [15] to accompany their explanation of execution state.

Images containing code snippets, output/errors, and autograder complaints – which together comprise 63% of images – were usually screenshots of the online code editor or text output window on the edX website. The fact that students resorted to *taking screenshots of plain-text content* to include in forum posts indicates that the forum's user interface for text entry is inadequate for their needs. One likely reason they took screenshots rather than simply copying-and-pasting text was to preserve spacing and indentation; some also drew annotations

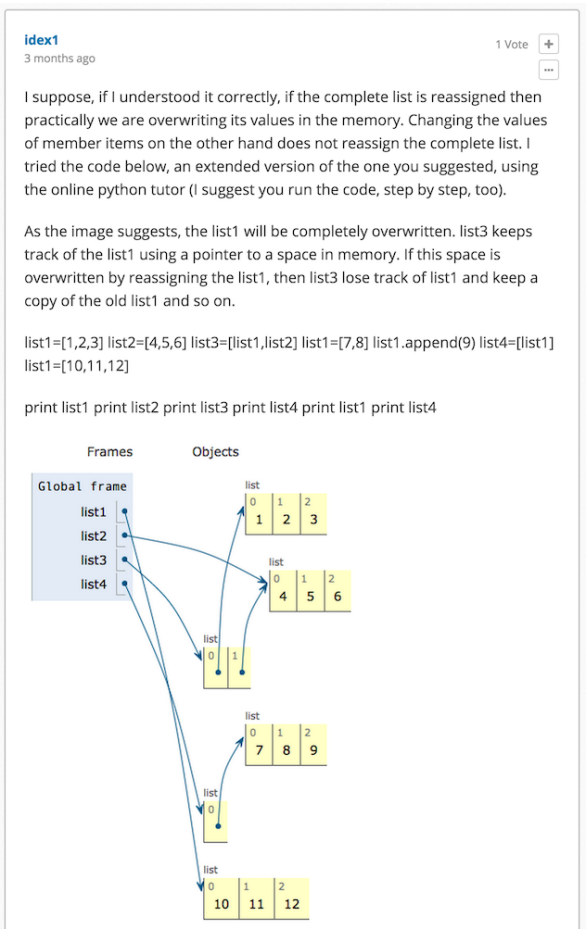


Fig. 2. A forum post with an inline image attachment that is a screenshot taken from an external program visualization tool (Online Python Tutor [15]). Also, note that the code mentioned in this post is not properly formatted.

atop the screenshots to highlight selected portions.

**Summary**: The majority of discussion forum threads in this course (∼60%) mentioned execution state, code snippets, output/errors, or autograder complaints. Thus, improving how these topics are rendered could have a noticeable impact on future forums for computer programming courses.

## V. Representative Examples of Post Topics

We now describe the most commonly-seen examples of each topic and summarize user frustrations with each.

### A. Execution state

Decades of computing education research have shown that developing a robust mental model of program execution is a fundamental skill for becoming a competent programmer [12], [13], [14]. This finding is supported by the fact that one of the most common kinds of discussions that occurred in the forum was about dynamic execution state – i.e., what happens "under the hood" as the computer executes a piece of code step by step. However, the plain-text format of the forum is not well-suited for holding conversations about the two major aspects of execution state: control flow and data structure values. Here are two representative posts about control flow:


Fig. 3. A post that shows code, program output, and an inline image attachment made using an external drawing tool.

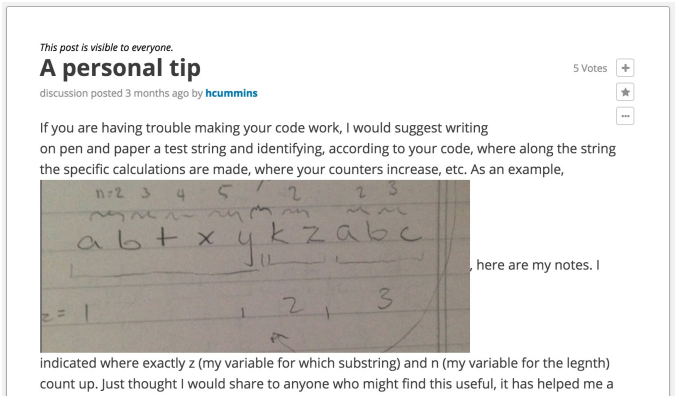


Fig. 4. A post with an image attachment that is a hand-drawn diagram.

"*Q1-3: I try the code and don't get an IndexException, therefore it should run the else clause, shouldn't it? So why it only prints the finally statement and then gives an error message? Q2-2: I get it that when it gets an IndexError it changes the function and run the code again. But why the answer is not 0, 1, 0 meaning that after changing the function it printed the finally and then printed the else and finally again? Q3-3: why don't I print 1 on the else clause after I pass the except clause?*"

"*I wonder if someone would be kind enough to explain the flow of control with a call to fib(x-1) and fib(x-2) in the same statement(expression?). Does the function call all of the (x-1) first and return those and then call all of the (x-2) values, or does it call both of them at the same time for the same value of n?*"

These kinds of posts are typically followed by a series of confusing replies where students try to articulate what they think is happening as the code executes. But there is no way for them to see and discuss what actually happens during execution, since the forum is not integrated with a debugger.

To discuss run-time values of data structures, students either tried describing how the data looks in their post or attached images containing screenshots taken from an automated program visualization tool (Figure 2), drawings from an illustration tool (Figure 3), or even pictures taken of hand-drawn sketches (Figure 4). Again, text is not well-suited for conveying concepts that are better conveyed as visualizations, but standard forums have no such visualization capabilities.

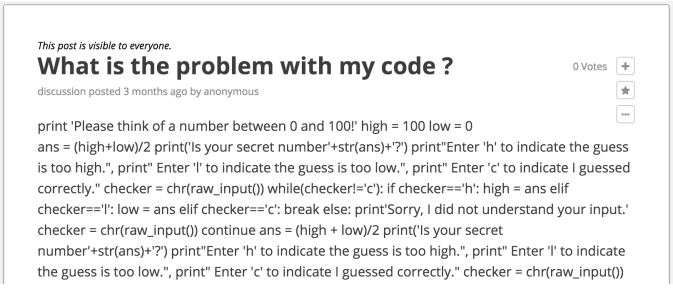


Fig. 5. A post showing a block of unformatted (presumably pasted) code.

**Summary**: Generic forums have no built-in way of visualizing execution state, so posters must resort to indirect textual explanations or screenshots from external applications.

### B. Code snippets

Students often copied and pasted code snippets into their posts (Table II), which is expected since this is a programming-intensive course. Although the forum had a "format-as-a-code-block" feature, many students did not know how to use it, so their code ended up looking like an unindented mess, as shown in Figure 5. Although this issue seems superficial, if someone's code is hard to comprehend, then others may be less likely to respond with useful help, or even to respond at all.

Students also interspersed code with accompanying explanations or questions, again without using special fonts to demarcate their code. Here is a short example: "*Why does type(varA) or type(varB) == str always evaluate to True?*" Although experienced programmers know how to parse these sentences, novices unfamiliar with Python syntax can have trouble telling which parts are code and which are English.

When someone is replying to a post containing code, they cannot directly *point to* specific parts like people can do if they are sitting in front of the same computer debugging together. So students referred to previously-posted code by copying and pasting it into their replies, and then making edits or adding comments. This behavior resulted in threads where the same code snippet was repeated multiple times with minor variations, which made it hard to hone in on meaningful diffs.

Presumably to avoid formatting issues, some students took screenshots of their code editor, drew arrows to highlight certain parts, and then pasted those images into posts. But doing so prevents others from copying and pasting the code into their replies, thus hindering further discussion.

**Summary**: Generic forums are not designed for holding rich conversations around source code; they treat code simply as plain text with some optional formatting.

### C. Output/errors

Students often copied and pasted outputs and error messages from program executions into their posts to ask about what went wrong. Just like with code snippets, many did not know how to use the "format-as-a-code-block" feature, which resulted in threads being littered with unformatted blobs of text. To avoid formatting problems, some students attached

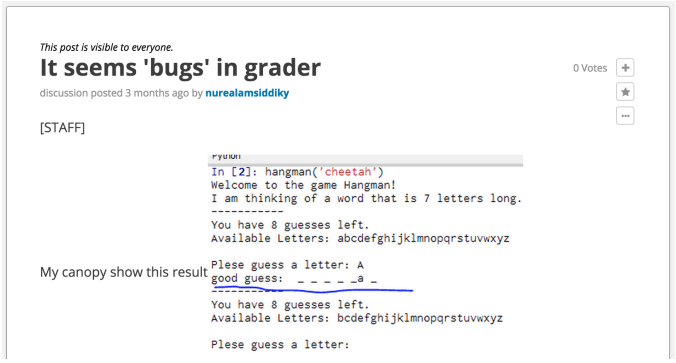


Fig. 6. A post complaining about output discrepancies with the autograder, using an attached screenshot with one part underlined in blue.

screenshots of their terminal output and highlighted specific lines (Figure 6).

Even if formatting were not an issue, the sheer volume of output and/or error messages dominated some threads. Since novices did not know which parts were significant, they simply copied and pasted everything in their terminal, which made it hard for others to read and refer to specific parts in their replies.

Discussions about output and errors were hampered by the fact that nobody could *reproduce the original executions* that created those outputs or edit the poster's original code to debug the underlying issues. All people could do was read long blobs of text and speculate on what they think happened in the original code to produce that text. In contrast, if people were sitting in front of the same computer debugging and experimenting together, they could directly edit the code and re-execute to see how their changes affect the output and errors.

**Summary**: Generic forums are not linked with an underlying code execution engine, so they cannot effectively capture discussions and experimentation around program output/errors.

### D. Autograder complaints

The final major topic we saw in the forum was complaints about the autograder software that checks the correctness of programming assignments. Since it is tedious to manually grade student assignments, many computer programming courses use some kind of autograder software. This course uses a standard type of autograder that compares the output of the student's code with known-correct outputs provided by the staff. However, it provides only a binary right-or-wrong response and does not give any feedback on *why* a student's answer may be wrong. The rigidity of the autograder led to many student complaints. These often arose from the student's output differing from the expected output in some minor way such as extra whitespace, misspellings, or punctuation marks. Since it was hard for students to see the diffs between their output and the expected output, they often accused the grader of having a bug since their own output "looks right." In these kinds of forum posts, students often copied and pasted large chunks of autograder output, similar to posting output/errors.

Figure 6 shows a typical complaint where a student posts a snippet of their output, argues for why it ought to be the correct output, and then accuses the grader of having a bug. This kind

of complaint was so pervasive that one of the Community TAs eventually replied: "*It's possible that there could be a problem with the grader. But so far it's been the student each time. I know it can be frustrating when it doesn't work and you believe the problem is with the grader. But you have to step back, take a deep breath, calm down a bit, and then post when you aren't angry. It will most likely save you the embarrassment of finding out that you were wrong, not the grader, after you've lost your patience and accused the grader of being wrong.*"

Aside from formatting gripes, a deeper source of frustration with the autograder was a lack of transparency in how it operated. Students often did not know which global variables and functions were pre-defined by the autograder, so they overrode those in their own code, which led to mysterious errors. Also, the runtime environment of the autograder differed from how students ran code on their own computers (e.g., differing versions of Python, operating systems, or locales), so code that worked locally sometimes failed in the autograder.

Just like how program visualizations expose the way execution state looks "under the hood," future forum designers could create similar visualizations to show how the grading process works step by step, so that students can see exactly how their work is being assessed. It can be aggravating to work for dozens of hours on a programming assignment only to have the grader simply print out a single word: "Wrong."

**Summary**: The autograder is a central component of many programming courses, but how it works is opaque to students. Generic forums provide no affordances to see how the autograder operates or to annotate its output.

## VI. Limitations of Study

We studied only one forum in one particular course, so replicating on additional courses would improve external validity. Note that this course is one of the largest and most popular MOOCs for introductory computer programming [11], and other programming MOOCs and even residential courses use the same kind of generic text-based forum as this one. Thus, hopefully our findings generalize to those settings.

Also, we did not personally interview students to see if they were truly frustrated with features of the forum's user interface. Rather, we inferred feelings of frustration based on our own judgment of the tone and content of student posts.

Our inter-rater reliability scores were only moderate. More iterative rounds of coding would have likely improved those scores, but the focus of this study was on qualitative aspects of forum limitations, so the accuracy of the exact numbers of posts in each topic does not change our main findings.

Finally, we did not perform any controlled experiments of different user interface conditions. This paper presents a purely retrospective study of a MOOC that has already completed. Thus, we cannot make any claims that features of the forum improve or hinder student learning in specific ways, since we did not formally test students' knowledge.

## VII. Toward a Domain-Specific Visual Discussion Forum for Learning Computer Programming

Our study has shown the ways in which the user interface of a generic forum is not ideal for topics that are commonly-discussed in introductory computer programming courses. We believe that a more effective forum should be *domain-specific* – catering to the needs of programming students – and also *visual* rather than purely text-based. We now propose one possible design for such a forum, based upon our study's findings.

**Extreme anchoring**: Our main idea is that all discussions should take place within the Web-based IDE (Integrated Development Environment) where students work on programming assignments. The IDE should contain these panes, which cover the most common student tasks and discussion topics:

- **Code editor**: standard editor with syntax highlighting
- **Output display**: emulates standard terminal output
- **Visual debugger**: single-step debugger that visualizes execution state such as stack frames, variables, data structures, and pointers (see Figure 7 for an example)
- **Autograder visualizations**: for more transparency, use the aforementioned visual debugger to also visualize the test environment in which the student's code is being run, including which functions and data are pre-defined, and how the grader executes their code to produce the final verdict of "right" or "wrong"

Educators have found *anchored discussions* [25], [26], [27] to be effective since discussions take place near the respective course content instead of in a separate forum. For our proposed design, we take anchoring to the extreme: students should be able to start discussions by selecting any element of the IDE and popping up a mini-discussion thread directly on top of it. ***Thus, students can annotate any line of code or output, and any data structure in the visual debugger***. That way, students no longer need to copy-and-paste code, output, and autograder messages into a separate disconnected forum and deal with the ensuing formatting gripes and lack of context (e.g., Figure 5).

Also, this IDE must fulfill all of a student's programming needs throughout the course, so that they do not need to install software or run code on their own computer. By providing a single run-time environment, we eliminate all forum questions about outputs being different on different students' computers.

**Synchronizing with a traditional forum**: Students spend most of their time in the edit-run cycle: editing code in the *code editor*, executing that code, studying the output and errors in the *output display*, and then re-editing. Let's say that a student Bob has a question about some portion of his code or output while he is working. Instead of copying-and-pasting into a traditional forum, Bob simply highlights the relevant snippet and starts an anchored discussion thread right on top of it.

How can anyone else see Bob's new thread, though? Every student is working in their own independent IDE on their own assignments, so they are not actively monitoring Bob's IDE for changes. To solve this problem, we propose to still keep a traditional forum in the course, but to have the IDE automatically post threads to that forum on the student's behalf. In the above example, when Bob highlights the code

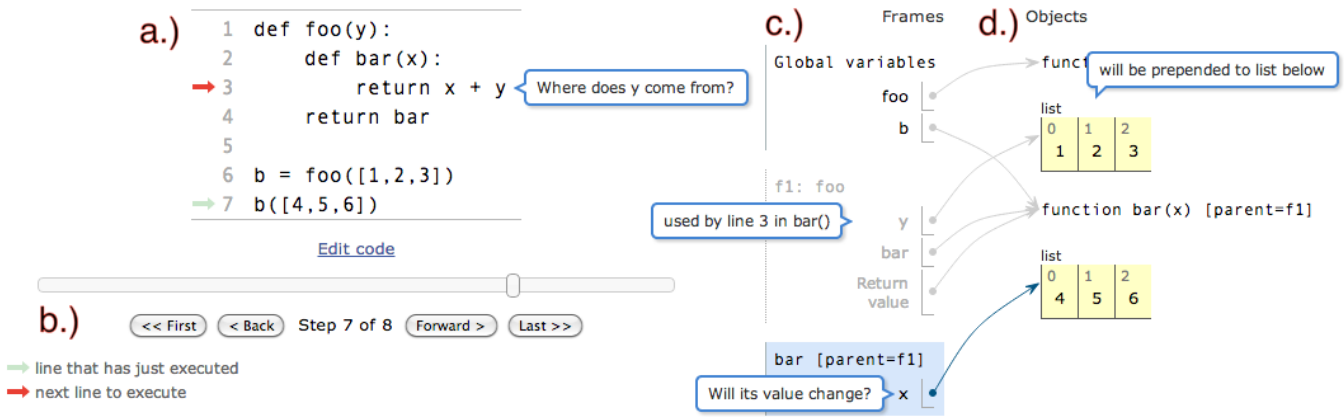


Fig. 7. A program visualization tool such as Online Python Tutor [15] enables the user to: a.) write code, b.) execute it and then single-step forwards and backwards through execution points, and see a visualization of c.) stack frames and d.) data structures at each step. We propose to augment this tool with annotation bubbles (shown in blue in this mock-up) so that users can click on any element in the code or visualization and start an anchored discussion about it.

snippet to start a discussion, the IDE posts a new thread for him, tagged with the current assignment he is working on and with the post's contents set to his question and code snippet.

It is important to keep the familiar format of a traditional forum since nearly 40% of topics in our course were *not* about code, execution, output, or the autograder (see Table II). We still want to allow students to discuss higher-level or miscellaneous topics on the forum. However, since the IDE makes it easy to start anchored discussions and auto-posts them to the main forum, we hope that students never need to copy-and-paste code or output, thus avoiding the mess of Figure 5.

**Code snapshots, versioning, and diffs**: After Bob highlights a snippet and starts an anchored discussion, the IDE posts it to the main forum. Now say that Alice is browsing the course's main forum and sees that new post from Bob. When she opens it, the forum redirects her to a *snapshot* of Bob's IDE that contains the exact version of code he posted, along with the anchored discussion thread within it. Note that this snapshot is probably not Bob's current code, since he has probably edited it since posting his question several minutes or hours earlier.

Unlike a normal forum post, this snapshot is not a static block of text; it is a live instance of the Web-based IDE. Thus, Alice can not only see Bob's code, but she can also execute it to see outputs and step through it with the visual debugger. Most importantly, she can edit Bob's code to try out different variations. If she modifies Bob's code and then posts a reply to his thread, the IDE saves a new version and records that it was derived ("forked") from Bob's original version. The IDE then automatically posts Alice's reply under Bob's original thread and shows a compact diff of her code versus Bob's to show what she has changed. Anyone can click on that diff to expand it and see a side-by-side view of Alice's versus Bob's code.

This snapshotting and auto-versioning feature makes it easy for people to comment on and modify each other's code without cluttering up forum threads by copying-and-pasting nearly-identical pieces of code. Instead, people can simply edit code in their IDE and have the system automatically create new snapshots that are linked to their respective posts. Finally, the IDE keeps track of the lineage (provenance) of code edits by all posters, so that Bob (the OP) can learn from how respondents have changed his code while trying to answer his question.

**Automatically-generated program visualizations**: Finally, recall that one of the most common kinds of discussions was about dynamic execution state, but the plain-text forum format makes it hard to converse about control flow and data structure values. As a workaround, some students have pasted images into their posts. To ease such discussions, we can integrate automatically-generated program visualizations into the forum.

Figure 7 shows a Web-based program visualization system called Online Python Tutor [15], which enables the user to: a.) write code, b.) execute it and then single-step forwards and backwards through execution points, and see a visualization of c.) stack frames and d.) data structures at each step. This tool can be integrated into the IDE so that students can use it as a *visual debugger* to diagnose what is wrong with their code. Also, it can serve as an *autograder visualization* to display how the autograder sets up the test environment and then steps through the student's code to produce the final grade.

While visualizing execution, the user can click on any line of code, stack frame, or data structure and start an anchored discussion on top of it (see blue pop-up bubbles in Figure 7). Using this mechanism, students can directly talk about each step of the visualization instead of attaching diagrams to regular forum posts (e.g., Figure 2). Thus, instead of trying to describe execution state in words, students can see their code's actual execution state and ask questions about it.

## VIII. Conclusion

We presented an empirical study of 5,377 discussion forum threads from a popular MOOC, *Introduction to Computer Science and Programming Using Python*, and found that around 60% of threads were about execution state, code, output/errors, or autograder complaints. We showed representative examples of such threads, which indicate how a generic text-based forum is cumbersome for discussing topics that are ubiquitous in computer programming classes. To ease such discussions, we proposed a new kind of domain-specific visual forum that integrates anchored discussions and program visualizations into a Web-based IDE. In future work, we hope to implement, deploy, and evaluate the pedagogical efficacy of our proposed design. Reducing friction in the user interface will hopefully enable programming students to hold more engaging discussions.

## REFERENCES


[1] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09. New York, NY, USA: ACM, 2009, pp. 1589–1598.

[2] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: Integrating web search into the development environment," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 513–522.

[3] A. Anderson, D. Huttenlocher, J. Kleinberg, and J. Leskovec, "Discovering value from community activity on focused question answering sites: A case study of stack overflow," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 850–858.

[4] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 2857–2866.

[5] J. Huang, A. Dasgupta, A. Ghosh, J. Manning, and M. Sanders, "Superposter behavior in MOOC forums," in *Proceedings of the First ACM Conference on Learning @ Scale Conference*, ser. L@S '14. New York, NY, USA: ACM, 2014, pp. 117–126.

[6] D. Coetzee, A. Fox, M. A. Hearst, and B. Hartmann, "Should your MOOC forum use a reputation system?" in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '14. New York, NY, USA: ACM, 2014, pp. 1176–1187.

[7] "Codecademy: Learn to code," http://www.codecademy.com/, accessed: April 2015.

[8] "Khan Academy: Computer programming," https://www.khanacademy.org/computing/computer-programming, accessed: April 2015.

[9] S. L. Emerson, "Usenet: A Bulletin Board for Unix Users," *Byte magazine*, vol. 8, no. 10, pp. 219–236, October 1983.

[10] H. Rheingold, *The virtual community : homesteading on the electronic frontier*. Reading, Massachusetts: Addison Wesley, 1993.

[11] "edX course: Introduction to Computer Science and Programming Using Python," https://www.edx.org/course/introduction-computer-science-mitx-6-00-1x-0, accessed: March 2015.

[12] B. Du Boulay, "Some difficulties of learning to program," *Jour. Educational Computing Research*, vol. 2, no. 1, 1986.

[13] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas, "A multi-national study of reading and tracing skills in novice programmers," in *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education*, ser. ITiCSE-WGR '04. New York, NY, USA: ACM, 2004, pp. 119–150.

[14] J. Sorva, "Visual program simulation in introductory programming education," Ph.D. Dissertation, Aalto University, 2012.

[15] P. J. Guo, "Online Python Tutor: Embeddable Web-based Program Visualization for CS Education," ser. SIGCSE '13. ACM, 2013, pp. 579–584.

[16] D. Rushkoff, *Program or Be Programmed: Ten Commands for a Digital Age*. Soft Skull Press, 2011.

[17] S. Y. Schoenebeck, "The secret life of online moms: Anonymity and disinhibition on youbemom.com." in *ICWSM*, E. Kiciman, N. B. Ellison, B. Hogan, P. Resnick, and I. Soboroff, Eds. The AAAI Press, 2013.

[18] Y. R. Tausczik, A. Kittur, and R. E. Kraut, "Collaborative problem solving: A study of mathoverflow," in *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW '14. New York, NY, USA: ACM, 2014, pp. 355–367.

[19] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman, "Knowledge sharing and yahoo answers: Everyone knows something," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 665–674.

[20] F. M. Harper, D. Raban, S. Rafaeli, and J. A. Konstan, "Predictors of answer quality in online q&a sites," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 865–874.

[21] S. Ahmed, S. Yang, and A. Johri, "Does online q&a activity vary based on topic: A comparison of technical and non-technical stack exchange forums," in *Proceedings of the Second (2015) ACM Conference on Learning @ Scale*, ser. L@S '15. New York, NY, USA: ACM, 2015, pp. 393–398.

[22] A. Furtado, N. Andrade, N. Oliveira, and F. Brasileiro, "Contributor profiles, their dynamics, and their importance in five q&a sites," in *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, ser. CSCW '13. New York, NY, USA: ACM, 2013, pp. 1237–1252.

[23] G. Wang, K. Gill, M. Mohanlal, H. Zheng, and B. Y. Zhao, "Wisdom in the social crowd: An analysis of Quora," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 1341–1352.

[24] M. S. Bernstein, A. Monroy-Hernández, D. Harry, P. André, K. Panovich, and G. G. Vargas, "4chan and/b: An analysis of anonymity and ephemerality in a large online community." in *ICWSM*, 2011.

[25] A. J. B. Brush, D. Bargeron, J. Grudin, A. Borning, and A. Gupta, "Supporting interaction outside of class: Anchored discussions vs. discussion boards," in *Proceedings of the Conference on Computer Support for Collaborative Learning: Foundations for a CSCL Community*, ser. CSCL '02. International Society of the Learning Sciences, 2002, pp. 425–434.

[26] M. Guzdial and J. Turns, "Effective discussion through a computer-mediated anchored forum," *Journal of the Learning Sciences*, vol. 9, no. 4, pp. 437–469, 2000.

[27] S. Zyto, D. Karger, M. Ackerman, and S. Mahajan, "Successful classroom deployment of a social document annotation system," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 1883–1892.

[28] S. Dewiyanti, S. Brand-Gruwel, W. Jochems, and N. J. Broers, "Students experiences with collaborative learning in asynchronous computer-supported collaborative learning environments," *Computers in Human Behavior*, vol. 23, no. 1, pp. 496 – 514, 2007.

[29] V. P. Dennen, "Pedagogical lurking: Student engagement in non-posting discussion behavior," *Comput. Hum. Behav.*, vol. 24, no. 4, pp. 1624–1633, Jul. 2008.

[30] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: Improving cooperation between developers and users," in *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '10. New York, NY, USA: ACM, 2010, pp. 301–310. [Online]. Available: http://doi.acm.org/10.1145/1718918.1718973

[31] A. J. Ko and P. K. Chilana, "Design, discussion, and dissent in open bug reports," in *Proceedings of the 2011 iConference*, ser. iConference '11. New York, NY, USA: ACM, 2011, pp. 106–113. [Online]. Available: http://doi.acm.org/10.1145/1940761.1940776

[32] O. Baysal, R. Holmes, and M. W. Godfrey, "No issue left behind: Reducing information overload in issue tracking," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014. New York, NY, USA: ACM, 2014, pp. 666–677. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635887

[33] P. K. Chilana, T. Grossman, and G. Fitzmaurice, "Modern software product support processes and the usage of multimedia formats," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '11. New York, NY, USA: ACM, 2011, pp. 3093–3102. [Online]. Available: http://doi.acm.org/10.1145/1978942.1979400

[34] "Course Philosophy (PDF): Introduction to Computer Science and Programming Using Python," https://courses.edx.org/c4x/MITx/6.00.1x_5/asset/6001x_course_philosophy.pdf, accessed: March 2015.