2023 Annual Conference & Exposition
Baltimore Convention Center, MD | June 25 - 28, 2023
The Harbor of Engineering Education for 130 Years
Paper ID #38722

Computing students' design preferences and barriers when solving short programming problems

Joseph Paul Hardin
Marc Diaz
Amanpreet Kapoor,

Amanpreet Kapoor is a lecturer in the Department of Engineering Education, and he teaches computing undergraduate courses in the Department of Computer & Information Science & Engineering (CISE). He received his M.S. in Computer Science from the U

Computing Students' Design Preferences and Barriers when Solving Short Programming Problems

Abstract

Modern educational assessment methods for computing that measure computer science (CS) students' learning of programming have increasingly moved to online or computer-based testing formats with recent advancements in educational technologies. Such methods to test a students' aptitude include browser-based applications where students receive immediate feedback on code, cloud-based autograders, online exams, and applications installed on remote virtual machines that provide students' feedback via testing libraries. With rising enrollment in computing, we need to ensure that assessment methods accommodate learners with different needs and preferences. This paper aims to describe the needs, preferences, and the barriers of students as they write short programs that provide them with feedback. Our data is collected through a survey which follows students' interaction with our web-based drill and practice programming system called Edugator in the context of a Data Structures and Algorithms (DSA) course at a large public university in the United States. Our system provided students two workflows for solving and receiving feedback on short programming problems: (1) using a browser-based workflow and/or (2) downloading an equivalent template of the problem and feedback, and coding it locally on their computer (a native workflow). We qualitatively coded 199 students' responses regarding choices using inductive thematic analysis to identify common themes. Our study found that while most students were motivated by convenience and ease of use to solve programming problems in browser-based systems, there is a specific cohort that prefers to program locally on full-fledged Integrated Development Environments (IDE) due to limitations in systems designed for hosting short programming problems such as lack of debuggers, students' familiarity with used IDEs, etc. Our research has implications for computing educators, system designers, and other stakeholders involved with the design or selection of programming environments or workflows who want to accommodate eclectic learner needs and support students to code natively or in browser-based solutions.

1. Introduction

Most computing courses require students to write short programming problems as assessments [1]. Although instructors who teach computing courses use eclectic tools and fragmented workflows to assess students' competence on coding skills, in the recent years there has been a rise in instructors opting for browser-based solutions that provide students immediate feedback on code correctness and other advanced code quality metrics. These solutions include:

(1) online judges such as DomJudge [2], Judge0 engine [3], Sphere engine [4], etc.,

(2) interactive textbooks that support code writing and testing such as Zybooks [5], Runestone Academy [6], Revel [7], etc.,

(3) code autograders such as Gradescope [8], Zylabs [5], Codegrade [9], CodeRunner [10], homegrown autograders, etc.
(4) cloud-based IDEs such as GitHub Codespaces [11], Replit [12], etc., and
(5) web-based platforms for competitive programming, technical interview practice, or coding practice such as Coding Bat [13], LeetCode [14], Codecademy [15], Codechef [16], Sphere Online Judge [4], Kattis [17], etc.

While there is substantial research on how these tools and platforms improve instructor productivity and aid in students' learning [18]–[20], we don't know much about what are student expectations from a system or workflows which allows them to practice short programming problems in the context of higher education. Our paper aims to understand student preferences and barriers on these workflows to ensure that our assessment methods accommodate undergraduate student learners with diverse needs and preferences.

In this paper, we aim to understand programming problem solving workflows post student interaction with Edugator that allows students to either use a web-based solution for coding and receiving feedback or solve the problem natively on their computer with the equivalent feedback through unit tests. Our platform was built iteratively using student feedback and it supports most features that are provided with other cloud-based solutions. We try to understand why some students prefer to work in a web-based solution while others code using an alternative native or local workflow. To understand these preferences and barriers, we designed a survey that students completed after interaction with our web-based drill and practice system in the context of a Data Structures and Algorithms (DSA) course at a large public university in the United States. We report on a qualitative analysis of 199 students' open-ended responses regarding students' choices. Our contributions from this paper are as follows: (1) rich descriptions of workflows for assessing and providing students feedback when solving short programming problems using cloud-based or native approaches and (2) student preferences and challenges when solving short programming problems using different mechanisms. These findings have design implications for computing educators, system designers, and other stakeholders involved with the selection of cloud-based or native programming workflows who want to accommodate eclectic learner needs and promote student-centered learning.

2. Prior Work

Studies on designing programming systems for novices include Pane and Myers' study [21] that reported multiple usability issues common in the features and design of "novice programming systems", outlining beneficial features as well as potential pitfalls in the several categories. These categories include visibility of system status (keeping the user informed about what is currently being executed in the system), match between the system and the real world (user friendliness, naturalness of the coding language and processes), user control and freedom (support for user notation, ease of code correction), consistency and standards (for example, usage of distinct syntax not easily confused), recognition over recall, aesthetic/minimalist design, error presentation, and documentation. These recommendations informed the design of our Edugator platform. For instance, when students press the "Submit" button on our system, they receive a cue, a circular progress bar, which signifies to the user the system status that their code is under execution and a feedback report would be generated. Other examples include the use of iconography that symbolized common metaphors for achieving tasks such as downloading files or resetting the problem, and consistency across the interface and the problem format for solving different programming problems. Pane [22] also touches on some of these previously mentioned concepts for designing systems for aiding children to program. Our work focuses on a different population, students enrolled in undergraduate degree programs.

Lahtinen et al. [23] elaborate on the roadblocks novice programmers face in a learning environment, including gauging subjects that programming students found most difficult, as well as uncovering what materials and study methods students found most helpful when constructing programs. Their work reported that students reported using a program development environment for learning to program, gaining access to computers, and finding bugs in their program hindered them to learn to program with ease. This work gives us insight into programming barriers, as does that of Ko et al. [24], who defined six learning barriers in the context of end-user programming. They identified six types of barriers related to design, selection, coordination, use, understanding, and information associated with end-user programming systems. Guo [25] describes additional barriers for a different population (non-English speakers) attempting to learn programming in an environment where most of the documentation and material is in English. He found that non-native English speakers faced barriers with instructional materials in English and suggested that designers should use simplified English and more visuals and multimedia for such populations. Jadud [26] described the processes novice programmers use when compiling programs in BlueJ and developed tools for visualizing these processes.

Prior work on short programming problems include work by Allen et al. [27] that found students in a CS1 course which used "many-small-programs" instead of a single large project were less stressed, more confident, had higher performance, and had higher satisfaction. There are a variety of platforms or instant feedback solutions used for hosting short programming problems with instantaneous feedback. These include Zylabs by Zybooks [5], Stepik [28], CodeRunner [10], CodeWorkout [29], [30], Leetcode [31], Runestone Academy [6], CodingBat [13], Codio [32], CloudCoder [20], etc. While we acknowledge that there are several open source and many proprietary platforms for hosting coding problems, as well as research on the effectiveness of these systems for teachers and students [18], there is not much research on students' preferences when interacting with the design or workflows associated with these systems. One paper that explored students' design preferences for online programming environments is by Olsson and Mozelius [33] who compared two online platforms - MyProgrammingLab and Codecademy - and found that students preferred unambiguous exercises, clear and well-formulated feedback, user friendliness, gamification and curriculum alignment for such platforms. Our work differs from Olsson and

Mozelius's work, as we contrast a native workflow with an online workflow instead of comparing two online platforms. We also suggest that instructors who may not have access to such platforms use unit tests to provide their students instant feedback.

3. Methods

3.1 Course and Institutional Context

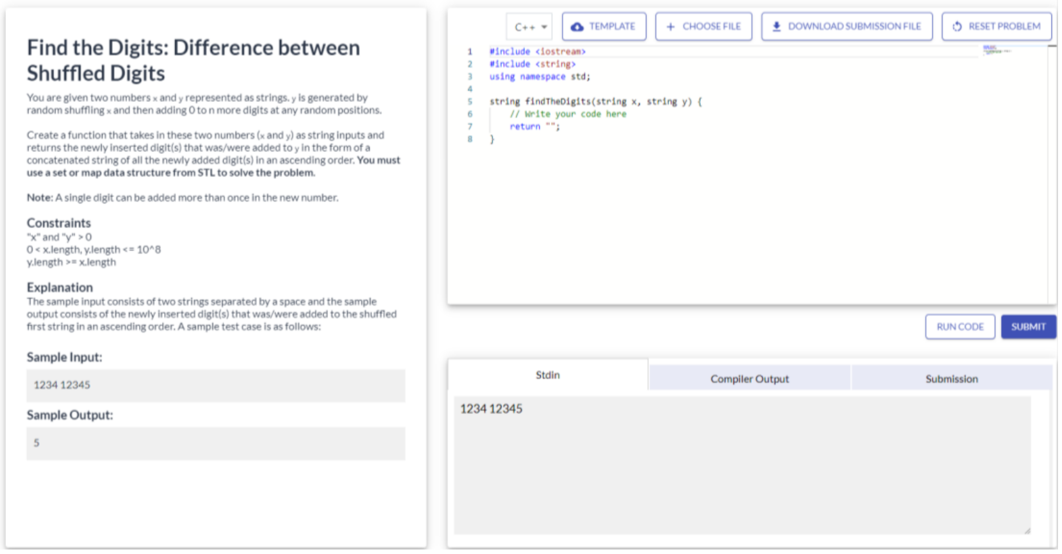
Our study was conducted in the Fall of 2021 in a Data Structures and Algorithms (DSA) course at a large public R1 university located in the southeastern United States. The course's pre- and co-requisites include two programming fundamentals courses (CS1 and CS2), a discrete mathematics course, and two calculus courses. Our DSA course covers (1) algorithm complexity analysis, (2) data structures including arrays, stacks, queues, linked lists, trees, sets and maps, and graphs, and (3) algorithm design techniques such as divide and conquer, greedy algorithms, and dynamic programming. Implementations of DSAs were covered in C++, and our graded assessments included two individual projects, a final self-proposed group project, two exams, thirteen weekly quizzes that included conceptual and programming questions, and several low stakes short programming problems for practice.

In this paper, we focus on data collected from a survey after students' interaction with two of our quizzes which were hosted on our drill and practice system which was under active development at the time of the study. These quizzes were (1) Quiz 9 where students were asked to write a C++ problem that used sets or maps to identify additional characters to a string that was randomly shuffled and added with more characters; and (2) Quiz 13 where students were asked to write a C++ problem to find a maximum sum path in a two dimensional container (maze) that required them to use dynamic programming (see Figure 1). We consider both of these questions as short programming questions as a typical solution would require less than 50 lines of code. The first author took this course previously as a student and the second author was the instructor of the course in Fall 2021. The third author is the technical project lead for the drill and practice system and was also an undergraduate teaching assistant for the course.

3.2 System Design and Assessment Interaction

Our Edugator platform was designed to host interactive programming exercises for computing courses. Students often face challenges to set up a development environment and it is often time consuming for Instructors to grade students' code and give feedback in the context of large classes. While solutions from vendors such as Zybooks, Codio, etc. exist in the market, they are often costly for the students and have to undergo institutional approval processes. We designed Edugator [34] as an open-source [35] platform that does not store any student data and does not require login. Edugator is a web-based application built using TypeScript, Redux, React.js, Material UI framework for frontend and uses a REST API for backend. The API was built using Node.js.

Figure 1: Description and design evolution of our system, Edugator across Quiz 9 and Quiz 13



Find the Digits: Difference between Shuffled Digits

You are given two numbers x and y represented as strings. y is generated by random shuffling x and then adding 0 to n more digits at any random positions.

Create a function that takes in these two numbers (x and y) as string inputs and returns the newly inserted digit(s) that was/were added to y in the form of a concatenated string of all the newly added digit(s) in an ascending order. **You must use a set or map data structure from STL to solve the problem.**

Note: A single digit can be added more than once in the new number.

Constraints
"x" and "y" > 0
0 < x.length, y.length <= 10^8
y.length >= x.length

Explanation
The sample input consists of two strings separated by a space and the sample output consists of the newly inserted digit(s) that was/were added to the shuffled first string in an ascending order. A sample test case is as follows:

Sample Input:

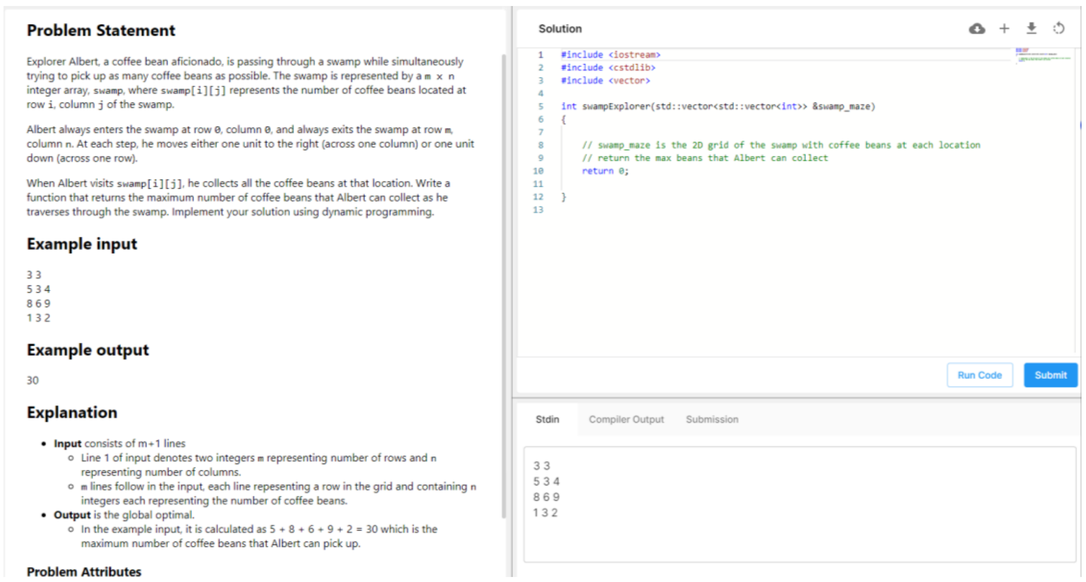
1234 12345

Sample Output:

5

```
#include <iostream>
#include <string>
using namespace std;

string findTheDigits(string x, string y) {
    // Write your code here
    return "";
}
```

1234 12345



Problem Statement

Explorer Albert, a coffee bean aficionado, is passing through a swamp while simultaneously trying to pick up as many coffee beans as possible. The swamp is represented by a `m x n` integer array, `swamp`, where `swamp[i][j]` represents the number of coffee beans located at row `i`, column `j` of the swamp.

Albert always enters the swamp at row `0`, column `0`, and always exits the swamp at row `m`, column `n`. At each step, he moves either one unit to the right (across one column) or one unit down (across one row).

When Albert visits `swamp[i][j]`, he collects all the coffee beans at that location. Write a function that returns the maximum number of coffee beans that Albert can collect as he traverses through the swamp. Implement your solution using dynamic programming.

Example input

3 3
5 3 4
8 6 9
1 3 2

Example output

30

Explanation

- **Input** consists of m+1 lines
  - Line 1 of input denotes two integers `m` representing number of rows and `n` representing number of columns.
  - `m` lines follow in the input, each line repesenting a row in the grid and containing `n` integers each representing the number of coffee beans.
- **Output** is the global optimal.
  - In the example input, it is calculated as 5 + 8 + 6 + 9 + 2 = 30 which is the maximum number of coffee beans that Albert can pick up.

Problem Attributes

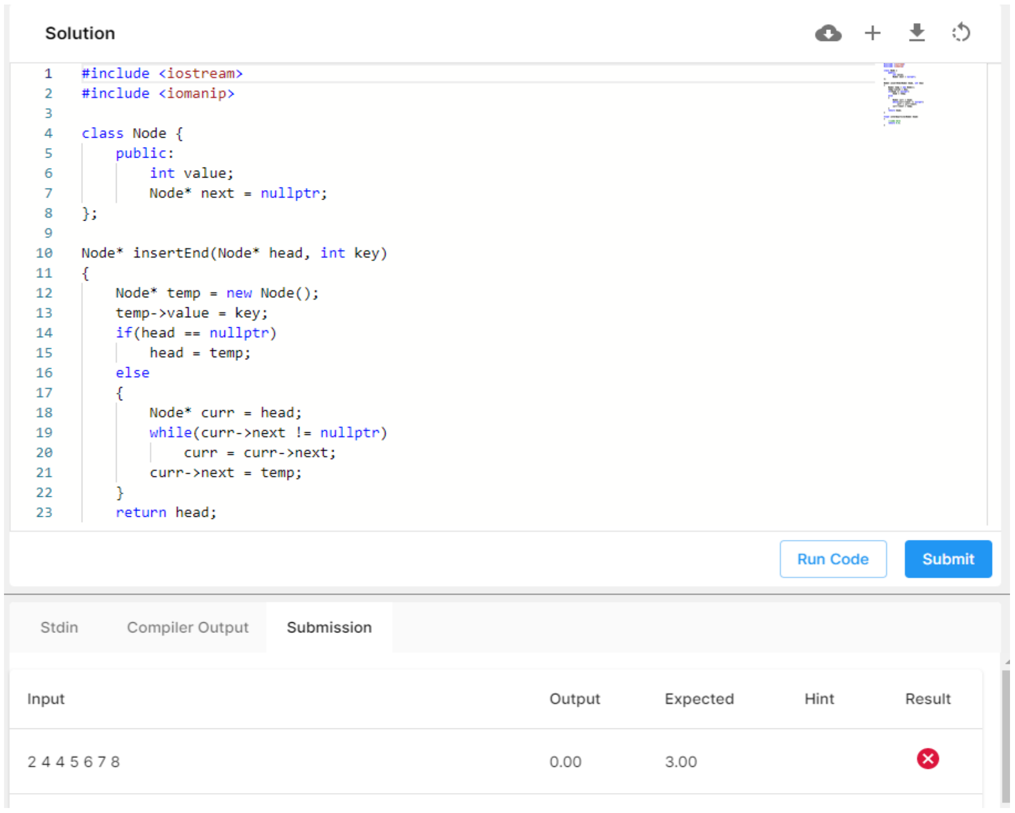
```
#include <iostream>
#include <cstdlib>
#include <vector>

int swampExplorer(std::vector<std::vector<int>> &swamp_maze)
{

    // swamp_maze is the 2D grid of the swamp with coffee beans at each location
    // return the max beans that Albert can collect
    return 0;

}
```

3 3
5 3 4
8 6 9
1 3 2

in Typescript with Express.js as the web app framework. We used another open-source code execution system called Judge0 to compile and execute students' coding submissions and Monaco text editor [36] for allowing students to write code with syntax highlighting and autocomplete in the browser. MongoDB is used to store each problem statement in the form of markdown, input and expected output based test cases, and other metadata. Architecturally, we utilized dependencies that are well documented and have a big community so that contributors can search for common problems with said dependency and the application was deployed on our university's server. The application is currently under active use and over 1800 students have used the application since 2021.

Features supported by our application include problem authoring and organization for instructors, and problem navigation, code writing or uploading local code files, building custom tests, testing against an instructor test suite, and downloading coded files for subsequent upload on learning management systems. Another feature our application supports for the students is offline delivery of corresponding problems along with test cases.

Figure 2: A sample problem with Input/Expected Output tests on our system, Edugator (web-based workflow) and the corresponding unit tests in a downloaded template (native workflow).



```
#include <iostream>
#include <iomanip>

class Node {
    public:
        int value;
        Node* next = nullptr;
};

Node* insertEnd(Node* head, int key)
{
    Node* temp = new Node();
    temp->value = key;
    if(head == nullptr)
        head = temp;
    else
    {
        Node* curr = head;
        while(curr->next != nullptr)
            curr = curr->next;
        curr->next = temp;
    }
    return head;
```

Input	Output	Expected	Hint	Result
2 4 4 5 6 7 8	0.00	3.00		✗

```
#include "../src/interquartile_range.h"
#include <vector>
#define CATCH_CONFIG_MAIN
#include "catch.hpp"

/*
    To check output:
        g++ -std=c++14 -Werror -Wuninitialized -o test test-unit/test.cpp && ./test
*/

TEST_CASE("Function: IQR 1", "[given]") {
    std::vector<int> v = {2, 4, 4, 5, 6, 7, 8};
    Node* head = nullptr;
    for(int i: v)
        head = insertEnd(head, i);

    REQUIRE(interQuartile(head) == 3.00);

    while (head != nullptr){
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}
```

Prior to our research survey, students could select one of two methods for coding a short programming problem which was given to them as a quiz question. The first option was to use a web-based text editor embedded in our application. Using Edugator, students could write, run, and receive correctness feedback on their code within their web browser without the need for downloads. Additionally, they could conduct simple I/O testing entering data in a text box that was redirected as standard input. The second option was to download a template from our drill and practice system. The template consisted of starter code which the students could use on their own machine (running locally) utilizing any Integrated Development Environment (IDE). This starter code was similar to the starter code provided in Option 1. The templates also had the accompanying unit tests written using the C++ Catch2 framework for the I/O tests shown in our Edugator platform (see Figure 2). Utilizing their own IDE, students could code locally, build tests, and conduct unit testing based on a provided template obtained from our Edugator application. Students also had an option to watch a pre-recorded video on how to navigate the two workflows and interact with our system before each quiz. After the quiz, students were given the option to opt-in to take our survey as described in the next subsection.

3.3 Study Design and Participants

We designed a survey to understand student needs when interacting with systems that provide them feedback when solving short programming questions. Data in this paper is a part of a study which was approved by the Institutional Review Board under the exempt category. Participation in the survey was voluntary and students received no incentive for their participation. Our goal was to use the knowledge from this study for iteratively refining the design of our drill and practice system and add features for supporting varied learner preferences. We accomplished this by means of a survey, which consisted of a set of questions listed in the next subsection. The survey was distributed twice, following two programming quizzes (Quiz 9 and Quiz 13), and feedback from the first survey informed us of design changes in our system before the second survey. 199 distinct undergraduate students enrolled in our DSA course consisting of 387 students took at least one of the two surveys and consented to use their data for research (Response rate: 51.4%). Overall, we received a total of 300 responses between both surveys. Students enrolled in our DSA course are primarily CS majors, followed by Computer Engineering majors, CS minors, Digital Arts and Sciences major, and double majors. We aim to answer the following research questions through our study:

1) What are undergraduate computing students' preferences for platforms that are used to support short programming problems?
2) What are the barriers which undergraduate computing students face when solving short programming questions on drill-and-practice systems?

3.4 Data Collection

Our survey consisted of three sections with skip logic for irrelevant sections based on a student's workflow for solving short programming problems. The landing page for the survey began with a question with three options: one option for those who solved the problem on their local IDE with a provided template, another option for those who solved the problem in the web browser using our system, and third if they followed another workflow. Depending on their answer to this question, the students were then guided through other questions that conformed to the option chosen. In total, the survey consisted of a maximum 5 open-ended questions and 12 multiple choice questions (MCQs) depending on a certain option. The questions were designed to elicit responses that were relevant for answering our research questions. Questions included inquiries on user experience (such as which components of the online or local workflows they used), why the student selected a particular workflow, desired features for our system, and open-ended feedback on the available options for solving short programming problems and receiving feedback. For this paper, we used the following two-open ended questions from our survey in our analysis:

(1) Why did you run your code on the Edugator and not use the template for running it locally? - [web-based workflow]
(2) Why did you use the template? - [native workflow]

3.5 Data Analysis

For analyzing the qualitative data, we used inductive thematic analysis [37] to identify trends in the open-ended responses regarding student experiences with writing solutions for short programming questions. Qualitative data was first categorized into codes and then abstracted to identify categories. From these categories, themes emerged (see Table 1). We also performed a frequency analysis on the categories and themes to shed light on learner preferences.

Table 1: Example of our Qualitative analysis

Survey Question: Why did you use the template?			
Raw Data	I was working with Edugator lost wifi and then lost all my work, so I decided to work with an IDE -*S137*	I had some issues that I needed a debugger for and the template made it relatively easy - *S153*	I am more comfortable with my IDE -*S167*
Primary Code	Wanting to avoid progress loss in Edugator platform	Wanting features not provided by Edugator	Personal preference for IDE
Category	Wanting to avoid shortcomings of Edugator		Preference against Edugator
Theme	Personal preference		

3.6 Limitations

Our data is from a small sample of the CS student population at our university from a single course, Data Structures and Algorithms. In addition, the data is from students who were using a third-party application at the time of the study for solving short programming problems as well as had used other workflows for solving programming problems in previous courses such as CS1 and Cs2. Their existing mental model and course workflows might have influenced their choices when interacting with our system. Data from a sample of CS students with no prior experience in programming might yield different results and hence our findings might not generalize to other contexts. Another limitation of our study is that data collected from surveys can induce response bias or interpretation of questions different from the intended meaning of a prompt. Lastly, data coded using qualitative analysis is subject to interpretation biases. We supplement our codes with representative quotes to increase validity and we are transparent about our coding process.

4. Findings

Our corpus consists of data from 199 students who interacted with our system and completed the survey after at least one of the two quizzes. In total we received 299 responses: 187 after students' interaction with the first quiz and 112 after the second quiz. 100 students completed both surveys and 99 completed one of the two surveys. The lower responses were possibly because the surveys were optional and there was no grade or monetary incentive to participate.

Of the 299 responses that we received for both quizzes, 71.2% student responses (n=213) mentioned using our Edugator platform for taking the quiz (web-based workflow), 16.4% responses (n=49) stated using a local IDE based native workflow, 10.4% responses (n=31) stated using both workflows, and 2% (n=2) stated using other mechanisms. Frequency analysis based on unique students indicates that 156 of the 199 students (78.4%) used the web-based workflow for at least one quiz, and 43 of the 199 students (21.6%) used the native IDE workflow for at least one quiz. 30 students stated that they used both web and local workflows and 6 students stated they used another workflow on a platform not used for hosting the quizzes as well as not using their local IDEs. This data forms the corpus of our qualitative analysis, and we focus on student interaction with web-based workflow or the native/local workflow. In total, we coded 49 responses that suggest interaction with the quiz using a native or IDE based workflow, and 203 responses which used a web-based workflow. The coded data was then analyzed to determine common themes in students' preferences and behavior in their coding practices. Each respondents' motivation for using their respective platform was assigned into one or more categories.

4.1. Students who used the Edugator Platform for Programming (Web-based workflow)

156 of the 199 students that participated in the use of the Edugator coding platform had many common themes underlying their responses. In the analysis of these responses, we were able to categorize the students' answers into six themes. The most common reason for students electing to use Edugator (web-based workflow) over their own IDE was convenience, with the majority of respondents (n = 140, N = 156, 89.7%) citing reasons relating to Edugator's (web-based workflow) ease of use and prevention of the inconvenience that the student has to go through to setup templates on their own machine. Other less common motivations for using Edugator (web-based workflow) were personal preference due to differences in speed, specific features, or other aspects

of operation in comparison to an IDE (n = 33, N = 156, 21.15%), believing that Edugator (web-based workflow) was sufficient for a coding problem of the caliber provided (n = 32, N = 156, 20.51%), wanting to try and experience the newly created platform (n = 14, N = 156, 8.97%), having to use the platform due to an inability to properly work with the template or lack of access to their own machine (n = 10, N = 156, 6.41%), or being unaware that there was another option for completing the quiz aside from Edugator (web-based workflow) (n = 4, N = 156, 2.56%).

As mentioned above, we coded our respondents' data based on recurring themes in their answers. To lend clarity, the following are the definitions of the themes used for those who participated in the use of the Edugator coding platform, as well as examples of respondents' answers to the question "*Why did you run your code on Edugator and not use the template*?" that were correspondingly assigned to these themes.

4.1.1. Preferences

- **Convenience** (140 respondents): These respondents elected to use Edugator (web-based workflow) on the grounds of convenience. Simply put, respondents thought the platform was easier and faster to use than to work on the alternative native/local workflow. Some respondents also shared that they felt lazy to set up the alternative native workflow.
  - *"Because it was already there to work on, and I was lazy" - S35.*
  - *"It was more convenient to do small tests on Edugator because the sample problems were already configured" - S5.*
  - *"The browser IDE was easier to use than downloading the template and doing it locally" - S69.*
  - *"Just because it was already nice and easy, I just had to click run!" - S9*
- **Personal Preference** (33 respondents): These respondents elected to use Edugator (web-based workflow) for a reason over the alternative, such as increased performance or additional functionality present in Edugator such as testing the solution with different inputs rather than building unit-tests for the native workflow or features supported by Edugator's text editor such as auto-fill.
  - *"Edugator has auto-fill functionality so no point in switching everything over to my ide" – S98.*
  - *"It was easier to submit and the check what cases I got right and which one I got wrong" – S194.*
  - *"It was more convenient for me to run code on the browser and it was easier to test my code with different sample inputs." – S1.*
- **Sufficient for small code problem** (32 respondents): These respondents elected to use Edugator (web-based workflow) due to the problem being, in their opinion, not complex enough to warrant coding outside of the Edugator text editor.
  - *"It was short enough to just do it on Edugator" – S74.*
  - *"Since it wasn't a super intensive program, I found it easier to code in browser*

rather than set up my local IDE for testing. If I ran into a difficult bug, I would have switched over to use a more powerful debugger." – S164.

  - *"If I had major bugs and needed the debugger I would have switched workflows and did it locally. I did it in edugator because I was able to start working eighth away, and being able to test inputs for their output seemed to be the only thing I needed for debugging." – S165.*
  - *"The problem was not complex enough that I needed to run/develop it on an IDE (no insidious bugs cropped up, nor were there enough "moving parts," so to speak, that I needed the debugging/developing bonuses yielded by Visual Studio or another bulky IDE). If this problem were on the level of complexity as, for example, the Gator AVL Project, I 100% would have downloaded the template and developed it locally. I also saw that the two test cases implemented in Catch on the template were the same two test cases that Edugator provided upon submission, so I had no need to download the template to test those two cases with catch. I also didn't have a need to write my own test cases with Catch due to the aforementioned simplicity of the problem; instead, I just changed the input in the "Stdin" section of Edugator for the few edge cases I could conjure up and they all passed on the first try, so I didn't really see a need to create numerous Catch tests. Once again, if this problem were far more complex, I would have downloaded the template and developed locally, utilizing many self-made Catch tests in the process". – S23.*

- **Desire to try format** (14 respondents): These respondents elected to use Edugator (web-based workflow) as a result of wanting to try the platform, be it because it was new, because it was similar to a technical interview platform, and more.
  - *"I wanted to test out the new environment" – S166.*
  - *"I figured that if Edugator was introduced for the first time, that I should at least try to make use of it" – S100.*
  - *"It looked very similar to leetcode and wanted to try it out" – S125.*
  - *"I think it is better to try out different interfaces especially to practice for coding interviews where I might be in an unfamiliar IDE". – S173.*

4.1.2. Barriers

- **Barriers to following alternate workflow - setting up or accessing local IDEs** (10 respondents): These respondents elected to use Edugator (web-based workflow) as a result of being unable to access or correctly operate the alternative option. These issues were mostly related to configuration in IDEs.
  - *"Was on a secondary machine, didn't have access to IDE. was able to run everything fine in browser so I went with it." – S196.*
  - *"I do not have a proper IDE installed" – S57.*
  - *"I was on my laptop which has no programs set up to use the template with" - S82.*

- **Unaware of options** (4 respondents): These respondents elected to use Edugator (web-based workflow) because they were not aware that there was another option by which they could complete the quiz.
    - *"I found it easier since I could also see the problem instructions. I also did not know we could run it on a template locally." – S24.*
    - *"Honestly, didn't know I could run it locally. I'm used to on quizzes just writing in the platform" – S152.*
    - *"Honestly, didn't notice the template, just went straight to solving the problem." – S97.*

4.2. Students who used a native IDE based workflow.

In the analysis of the responses of the 43 students who participated in the use of the provided template on their own machines and IDE, we were able to categorize the students' answers into four themes. These students also had a distinct most common reason for use of templates over the provided Edugator platform. This reason was personal preference due to familiarity with local IDE or speed of solving problems locally. 28 of the 43 respondents who used templates (65.12%) cited the former reason as their motivation for using native workflows. Other less common motivations for using native workflow were the desire to use features present in their own IDE that were not present in Edugator (n = 16, N = 43, 37.21%), having to use the templates due to unawareness of other options (n = 6, N = 43, 13.95%), and barriers to solving the question on Edugator (n = 3, N = 43, 6.98%). There were also respondents who, despite taking the survey and attesting that they chose the template method for solving the coding problem, did not use it (n = 5, N = 43, 11.63%). These are the definitions of the themes used for those who participated in the use of the template, as well as examples of respondents' answers to the question "*Why did you use the template*?".

4.2.1. Preferences

- **Personal Preference** (28 respondents) - These respondents elected to use the template/their own IDE for a subjective reason such as increased performance, and familiarity with the local workflow.
    - *"I am more comfortable with my IDE" – S167.*
    - *"I wanted to run the code in an environment I am familiar with and be able to use my debugger" - S131.*
- **Use of features not in Edugator platform** (16 respondents) - These respondents elected to use the template/their own IDE due to lack of features supported by Edugator and/or features that were available in their native workflows. Common features that were missing included code collapse, lack of debuggers, use of extensions and autocompletes, error messages from their IDEs that are more detailed.
    - *"I had some issues that I needed a debugger for and the template made it*

    - *relatively easy" - S153.*
    - *"I like to see error messages in my IDE (spelling errors, missing include statements, etc.)" - S163.*
    - *"Because I wanted to speed up the compiling/running process. Also, my IDE has features that make my coding experience easier and more fun such as colors, auto correct, and minimizing chunks of code." - S158.*

4.2.2 Barriers

- **Unaware of options** (6 respondents) - These respondents elected to use native workflows as a result of being unaware of the alternative option.
    - *"because it was needed for the quiz." – S146.*
    - *"thought we were supposed to" - S93.*
- **Barriers to following alternate workflow – using web-based workflow** (3 respondents) - These respondents elected to use the template/their own IDE as a result of barriers they faced or heard of when solving problems on Edugator. These barriers pertained to either automatic saving of their progress or VPN issues. At the time of the study, the application was deployed on our university's server, and it was available on the university network internally. However, this issue is fixed in the new version of Edugator.
    - *"I was working with Edugator lost wifi and then lost all my work, so I decided to work with an IDE" - S137.*
    - *"My IDE is faster, has a better autocomplete, better error detection, and i feel more comfortable being able to save the file locally in case power goes out or something like that" - S56.*
    - *"I wanted to be able to work on the quiz from home without having to stay connected to the VPN." – S75.*

4.3. Students who used a combination of both local and web-based workflows

A third group of individuals that we deemed worthy of note were students that elected to use a combination of multiple different workflows (selecting both their own IDE and the Edugator to utilize elements of both environments, n = 30). Their sentiments echoed those of the individuals who only used one of the platforms, such as wanting to use features in their IDE that were not present in Edugator, while also wanting to test and submit their code in Edugator. Therefore, they had similar preferences that were found in the previous two groups (e.g., 'Desire to try format' and 'Use of features not in Edugator the platform).

5. Discussion

Our findings suggest that most students are driven by convenience and ease of use of the platform when solving short programming problems. Other factors that influence their decision making when selecting either a native workflow or the web-based workflow include platform simplicity, support of features that enable them to answer a problem effectively, and familiarity with an accustomed workflow. This shows that, in providing new platforms, unless the platform is their first, this demographic will often fall back on a system which matches their mental model and uses design metaphors in line with other systems they have previously used, regardless of if the option they choose is objectively inferior. This factor is important to note in system development - most students will be more likely to opt into using a system that resembles one they have used before, as opposed to having to ‘reinvent the wheel’ and develop a new mental model which requires them to learn a new platform on top of achieving the core task at hand, solving a programming problem.

Another item of note from the data is the students’ analyses on the difficulty of the task placed on them in their decisions between platforms. The degree to which students attributed the short nature of the provided programming problems to their decision making indicates that, in designing or providing such systems in an educational environment, it is important to consider the intensity of the tasks that the target users will be using. The results of this study indicate, for example, that if they deem the work, they will have to do in such a platform to be manageable, a great proportion of students will elect to utilize a platform that may be more bare-bones than another. Conversely, these same students may elect to work in a more fleshed-out and feature-heavy environment if they are provided with a more extensive assignment or project.

In tandem with these findings, it is important to note the rationale that emerged for those students who elected to use their own IDE instead of the Edugator drill-and-practice system: the use of features not present in the web-based environment. Several students who opted for IDEs highlighted specific tools and elements that they enjoyed using in their own environment that the Edugator system lacked, such as debuggers, syntax highlighting, code chunk collapsing, no need for reliable internet, etc. This would suggest that, in combination with the aforementioned points regarding the difficulty of provided tasks, a system should include enough features to support the required tasks but should not under- or over-provide to the point where it infringes on functionality or the user experience.

6. Conclusion

To sum up, we recommend instructors to select platforms that accommodate learner needs based on the programming problem complexity and ease of use. In addition, we recommend instructors offer alternate means to enable students to program on their IDEs and deliver feedback to students who may not prefer to use web-based systems based on lack of features that are supported by more complex programming environments. The latter will also equip students with experience using more authentic tools that are used in the industry through repeated practice. For system designers, we recommend using popular design frameworks as well as design metaphors that match a student’s existing mental model and provide a user experience which matches their beliefs, minimizes learnability, improves discoverability of features through signifiers, and reduces students' cognitive load when interacting with a system. We also recommend that system designers provide alternate means to students who prefer not to use the web-based solutions to accommodate eclectic learners.

References

[1] C. L. Gordon, R. Lysecky, and F. Vahid, "The Rise of Program Auto-grading in Introductory CS Courses: A Case Study of zyLabs," in *2021 ASEE Virtual Annual Conference Content Access*, Jul. 2021. Accessed: Feb. 28, 2023. [Online]. Available: https://peer.asee.org/37887

[2] "DOMjudge - Programming Contest Jury System." https://www.domjudge.org/ (accessed Feb. 28, 2023).

[3] "Judge0 - Where code happens." https://judge0.com/ (accessed Feb. 28, 2023).

[4] "Sphere Online Judge (SPOJ)." https://www.spoj.com/ (accessed Feb. 28, 2023).

[5] "zyBooks - Build Confidence and Save Time With Interactive Textbooks." https://www.zybooks.com/ (accessed Feb. 28, 2023).

[6] "Runestone Academy." https://runestone.academy/ (accessed Feb. 28, 2023).

[7] "Revel | Pearson." https://www.pearsonhighered.com/revel/index.html (accessed Feb. 28, 2023).

[8] "Gradescope." https://www.gradescope.com/ (accessed Feb. 28, 2023).

[9] "CodeGrade - Virtual Assistant for your coding classroom." https://www.codegrade.com/ (accessed Feb. 28, 2023).

[10] "CodeRunner." https://coderunner.org.nz/ (accessed Feb. 28, 2023).

[11] "GitHub Codespaces." https://github.com/features/codespaces (accessed Feb. 28, 2023).

[12] "Replit: the collaborative browser based IDE - Replit." https://replit.com/ (accessed Feb. 28, 2023).

[13] "CodingBat Java." https://codingbat.com/java (accessed Feb. 28, 2023).

[14] "LeetCode - The World's Leading Online Programming Learning Platform." https://leetcode.com/ (accessed Aug. 31, 2019).

[15] "Learn to Code - for Free | Codecademy." https://www.codecademy.com/ (accessed Feb. 28, 2023).

[16] "Competitive Programming | Participate & Learn | CodeChef." https://www.codechef.com/ (accessed Feb. 28, 2023).

[17] "Kattis, Kattis." https://open.kattis.com/ (accessed Feb. 28, 2023).

[18] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä, "Review of Recent Systems for Automatic Assessment of Programming Assignments," in *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, 2010, pp. 86–93. doi: 10.1145/1930464.1930480.

[19] P. Ihantola *et al.*, "Educational Data Mining and Learning Analytics in Programming: Literature Review and Case Studies," in *Proceedings of the 2015 ITiCSE on Working Group Reports*, 2015, pp. 41–63. doi: 10.1145/2858796.2858798.

[20] J. Spacco *et al.*, “Analyzing Student Work Patterns Using Programming Exercise Data,” in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, 2015, pp. 18–23. doi: 10.1145/2676723.2677297.

[21] J. F. Pane and B. A. Myers, “Usability Issues in the Design of Novice Programming Systems,” Carnegie Mellon University, School of Computer Science, Technical Report CMU-CS-96-132, Pittsburgh, PA., 1996.

[22] J. F. Pane, “Designing a Programming System for Children with a Focus on Usability,” in *CHI 98 Conference Summary on Human Factors in Computing Systems*, 1998, pp. 62–63. doi: 10.1145/286498.286530.

[23] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, “A Study of the Difficulties of Novice Programmers,” in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 2005, pp. 14–18. doi: 10.1145/1067445.1067453.

[24] A. J. Ko, B. A. Myers, and H. H. Aung, “Six Learning Barriers in End-User Programming Systems,” in *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 2004, pp. 199–206. doi: 10.1109/VLHCC.2004.47.

[25] P. J. Guo, “Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–14. doi: 10.1145/3173574.3173970.

[26] M. C. Jadud, “An exploration of novice compilation behaviour in BlueJ,” *Comput. Sci. Educ.*, vol. 15:1, pp. 25–40, 2005, doi: 10.1080/08993400500056530.

[27] J. M. Allen, F. Vahid, A. Edgcomb, K. Downey, and K. Miller, “An Analysis of Using Many Small Programs in CS1,” in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 585–591. doi: 10.1145/3287324.3287466.

[28] “Stepik.” https://stepik.org/ (accessed Feb. 28, 2023).

[29] S. H. Edwards and K. P. Murali, “CodeWorkout: Short Programming Exercises with Built-in Data Collection,” in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*, 2017, pp. 188–193. doi: 10.1145/3059009.3059055.

[30] “CodeWorkout.” https://codeworkout.cs.vt.edu/ (accessed Feb. 28, 2023).

[31] “LeetCode .” https://leetcode.com/ (accessed Feb. 28, 2023).

[32] “Codio | The Hands-On Platform for Computing & Tech Skills Education.” https://www.codio.com/ (accessed Feb. 28, 2023).

[33] M. Olsson and P. Mozelius, “On design of online learning environments for programming education,” in *Proceedings of the 15th European Conference on e-Learning (ECEL 2016)*, 2016, pp. 533–539.

[34] “Edugator.” Accessed: Apr. 30, 2023. [Online]. Available: https://edugator.app/

[35] “Edugator.” https://github.com/edugator-cise (accessed Apr. 30, 2023).

[36] “Monaco Editor.” https://microsoft.github.io/monaco-editor/

[37] V. Braun and V. Clarke, “Using thematic analysis in psychology,” *Qual. Res. Psychol.*, vol. 3, no. 2, pp. 77–101, 2006, doi: 10.1191/1478088706qp063oa.