Student Adoption and Perceptions of a Web Integrated Development Environment

An Experience Report

Martin Velez, Michael Yen, Mathew Le
UC Davis, CA, USA

Zhendong Su
ETH Zurich, Switzerland

Mohammad Amin Alipour
University of Houston, TX, USA

ABSTRACT

Students often spend a considerable amount of time and effort installing and configuring programming tools and environments. This can frustrate, and distract them from more important learning objectives, particularly in introductory programming courses. A web-based integrated development environment can serve as a low-threshold, ready-to-use programming environment, and reduce the time and effort needed to start practicing programming.

In this paper, we report our experience of developing and deploying a web-based integrated development environment (web IDE) as an optional tool at a large public university that has been in use for over several years in various programming courses.

We conducted a survey to understand students' perceptions toward the web IDE and usage of its features. Using the data from the survey, we explored potential correlations between student demographic and behavioral traits in adoption of the web IDE. The results of the survey suggest that around half of the students use the IDE often or very often. We also discovered that the likelihood of adoption of the IDE decreases as students to move to upper classes. In this paper, we also describe broader lessons for educators and researchers.

CCS CONCEPTS

• **Applied computing** → **Computer-assisted instruction**; • **General and reference** → *Empirical studies.*

KEYWORDS

Web-based Integrated Development Envioronment; Student Perceptions; Tool Adoption

ACM Reference Format:
Martin Velez, Michael Yen, Mathew Le, Zhendong Su, and Mohammad Amin Alipour. 2020. Student Adoption and Perceptions of a Web Integrated Development Environment: An Experience Report. In *The 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20), March 11–14, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3328778.3366949

SIGCSE '20, March 11–14, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6793-6/20/03…$15.00
https://doi.org/10.1145/3328778.3366949

1 INTRODUCTION

Programming involves several steps such as coding, compiling, linking, testing and debugging [17]. Each step requires proper installation and configuration of corresponding tools and environments. Integrated development environments (IDE) attempt to integrate and present all tools needed for programming in a unified interface, but still they require users to configure the system and install individual tools. For example, an IDE for Python programming language will provide syntax highlighting for Python programs, but users may still need to install the Python interpreters and configure the IDE to use the right installation (*e.g.* python 2.7 or 3.6).

Installing and configuring programming tools and environments can be a frustrating and error-prone task, especially for a student that is learning programming, and may also distract the student from the primary learning objectives [15]. For example, at the University of California at Davis (UC Davis), there is an upper division "Programming Languages" course where students learn programming language theory and concepts by studying different languages, namely `C++`, Java, Lisp, and Prolog. Since the course is only ten weeks long (due to the quarter system), the assigned programming projects are limited in length and difficulty. Nonetheless, students have reported spending a significant portion of their time and effort (hours, even days) installing and configuring programming tools for each new assignment.

A web-based integrated development environment (web IDE) can provide a uniform, simple programming interface and require no installation or configurations on the local machine [10, 15, 30]. A web IDE is particularly desirable in classrooms because (1) it can reduce time for installation, configuration and troubleshooting of tools and environments, allowing students and instructors to focus on the primary learning objectives, and (2) it can provide a unified, reproducible execution environment, which can improve communicating programming problems between students and teaching staff [15].

We developed and deployed Kodethon, a web IDE at UC Davis. At the time of writing, we have operated and maintained Kodethon for more than three years. Kodethon supports all of the programming languages used to teach courses at the university, including Python, C, C++, Java, Lisp, and Prolog. Its built-in editor provides convenient features like syntax highlighting and auto-completion. Student can also use an easy-to-use shell or a full Unix terminal—within the browser—to execute Unix commands, such as, `ssh` and `scp`. Kodethon also supports real-time collaboration which facilitates pair-programming and live teaching assistance.

In this paper, we discuss adoption of the web IDE by students, and their perceptions toward it. Kodethon has been used by more

than 3,000 students in 15 courses as an *optional tool*, as opposed to previous studies, e.g., [3, 4], wherein students were *required* to use the pedagogical IDE. Thus far, students have used Kodethon to write more than 15 million lines of code. We analyze survey responses from 140 students who took a course recently to understand student usage and perceptions of Kodethon. The results suggest that 48% of survey participants use the web IDE often. Around a third of students agree about the usefulness of Kodethon, according to usefulness criteria outlined by Lund [21], About a third of participants opted to use the web IDE to write their programs. Students ranked "Web-based" and "No Installation Required" as the two most useful features – consistent with the premise of web IDEs as a low-threshold, ready-to-use programming environment. We found that class standing has a strong correlation with adopting the web IDE, as novice students are more likely to adopt the system. We found students that use the web IDE more often tend to not use the alternative stand-alone IDEs or editors. However, we did not find strong correlations between adoption of the web IDE and adoption of authoring web applications like Google Docs.

The main contributions of this paper are:

- We describe a web IDE that has had widespread use across diverse set of computer science courses as an optional tool (§3).
- We designed and conducted a user survey of student usage and perceptions of the IDE, and analyzed the results (§4).
- We explored correlations between adoption of a web IDE and student characteristics (§5).
- We discuss broader lessons for CS educators and researchers regarding web IDEs (§6).

2 RELATED WORK

Web IDEs for classrooms: Perhaps, the work most related to ours is a recent study by Benotti *et al.* that describes a web IDE to support teaching functional programming in Haskell, and evaluated the students' attitude toward it [4]. In their study, students strongly agreed that the web IDE makes them more productive, and a better Haskell programmer. Barr et al. developed CodeLab as a Web IDE for introductory programming courses. They observed that the average grade of students increased moderately after adopting CodeLab (0.2 points in a 4-scale grading) [3]. Note that in these studies students were *required* to use the web IDE for programming while we did not requires students to use Kodethon.

There are a number of web IDEs for classrooms. For example, PythonTutor is a popular Python tutoring system that in addition to execution of single-file programs, visualizes the heap of programs [11] . Helminen et al. developed a web IDE for Python programming and ran a user study [12]. In their study, more than 40% of students reported that they used the system frequently, and a large portion of students indicated that the web IDE is useful and should be used in future course offerings.

Pedagogical stand-alone IDEs: There are a number of tools, for example, to teach object-oriented concepts, such as classes and relationships between them. BlueJ IDE provides visual tools to design a class and define the relationships between a class and other classes [19]. BlueJ has been extended to be used in data structure courses [25], to accommodate collaboration [8], to teach design patterns [24], and in programming embedded systems [2]. A survey of

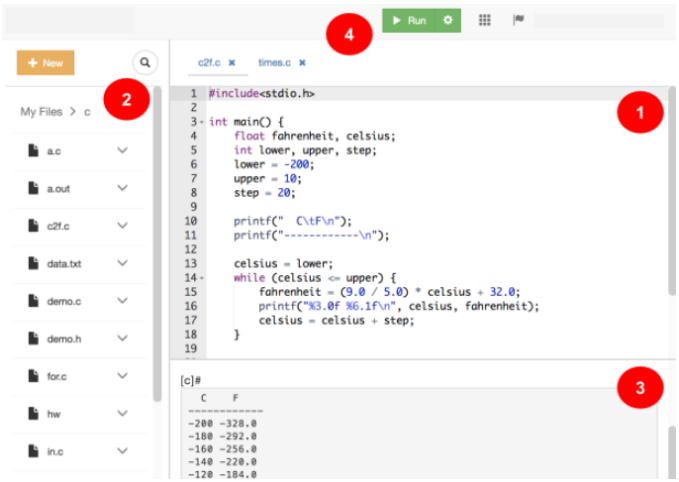

Figure 1: Kodethon User Interface: File navigator, editor, and CDE Shell.

computing education research community in 2006 shows that more than 25% respondents use BlueJ IDE in introductory courses [27].

Feedback in IDEs: In addition to editing and building programs, IDEs can also provide hints to assist students to improve their programming skills. For example, ASIDE [35], and ESIDE [34] nudges students to adopt secure programming practices, or DevEventTracker [16] evaluates how students adhere to principle of incremental programming and how much they procrastinate. Some IDEs also track students' activities to identify at-risk students [7, 9, 23].

Web IDEs in MOOCs: With the advent of MOOC and large scale learning, web IDEs are becoming commonplace. For example, Khan Academy provides a simple web IDE in their programming lessons [1]. Some web IDEs can offer automated feedback to students to fix the errors [31, 32]. Web-based development environments also serve as a platform for massive collection of data from users actions. The data can be used to develop various predictive models. For example, Wang et al. use data collected from EdX's web IDE to predict the success of students in arriving at a correct code, given the intermediate steps they take [33].

3 SYSTEM DESCRIPTION AND ADOPTION

Kodethon is a web integrated development environment—it requires no installation—and users can immediately start writing, executing, and storing programs. It is designed to be easy-to-use by university students, and to be useful in completing their coursework. To scale to classes of hundreds of students, we designed Kodethon as a distributed system that can scale horizontally.

3.1 Main Features

Figure 1 shows the main user interface which consists of an editor, a file navigator, a CDE shell, and a smart run button.

(1) *Editor:* The editor is an instance of the open-source ACE editor which provides syntax highlighting for multiple programming languages and basic auto-completion. Users can personalize the editor by changing the theme, font, and indentation settings. ACE even supports editing in VIM mode which some users find

to be a more efficient mode of editing. We added support to open multiple files and switch between them using tabs.

(2) *File Navigator:* Users can create files and organize files in folders. They also can rename, move, copy, upload, and download files. This allows students to access their files from anywhere and from any device.

(3) *Shell Environment:* This is a KODETHON-specific shell that allows users to see the output of the progrograms, and run common Linux commands like `ls` and `cd` but also allows users to run KODETHON-specific commands like `terminal` which opens a standard Linux terminal for more advanced users. The user can also compile and run programs using this shell.

(4) *Smart Run Button:* A common point of initial confusion for students is: "How do I run this program?" KODETHON employs a best-effort strategy for executing programs. For example, if the file end in `.c`, it will search for a `Makefile` and if it finds one, then it runs `make`. Otherwise, it will compile the current file, and run the resulting `a.out` program. Another example, if the file ends in `.l`, it will interpret it using `clisp`. In our experience, this helps students interact with programs much quicker. Users can customize build and run settings.

KODETHON also provides:

- *Multi-language Support:* It provides over a dozen programming environments, distinct sets of compilers, interpreters, and programming tools. For example, to program in Lisp, a user simply selects the `lisp` environment which includes the `clisp` interpreter. The environments are defined as Docker images. All user commands are executed in Docker containers. For scalability and performance, we limit resources per user such as disk space, RAM, and CPU usage.
- *Real-time Collaboration:* By default, a user starts with an always-private *project*. To collaborate with others, he/she must create a *shared project* and add collaborators. All collaborators will have access to that project. KODETHON allows users to set read/write/execute permissions for collaborators (think Linux groups), and for the public. We have observed that some users create a project for each homework while some create a project for the entire course. In a shared project, two or more people can edit the same file at the same time.
- *Learning management system:* Instructors create programming assignments. Students upload submissions to KODETHON. The submissions are automatically graded. Students receive feedback on their submissions by running the provided test suite.

3.2 Deployment and Adoption

To deploy KODETHON to students, we asked instructors who were teaching programming-intensive courses for the opportunity to give a live demonstration of KODETHON during one of their lectures. We gave our first live demo to students taking "Introduction to Programming", which uses Python programming language, in the Spring 2015. Since then, we have given more than ten live demos. Professors usually allot 5 to 15 minutes at the beginning of a lecture in the first or second week of the academic quarter. We usually demonstrate 1) how to create and edit programs, 2) how to execute programs with the smart button, 3) how to use the shell environemnt, 4) how to collaborate in real-time, and 5) how to access the more advanced but traditional Unix terminal. The programming language we used in the demo depends on the main programming language of the class. For a class like "Programming Languages", where students have to code in Java, Lisp, and Prolog, we emphasized the ease with which students can switch to and between the appropriate programming environments.

Table 1: Responses to the survey question: *"I use or have used KODETHON to do coursework in the following courses."*

#	Title (Intervention(s))	Programming Language(s)
1	Introduction to Programming (LD)	Python
2	Introduction to Programming (LD)	C
3	Software Development and Object-Oriented Programming **(LD, S)**	C++, Rust
4	Computer Organization and Machine-Dependent Programming **(LD, S)**	Assembly, C++
5	Data Structures and Programming (LD)	C, C++
6	Theory of Computation	N/A
7	Algorithm Design & Analysis	C, C++
8	Probability and Statistical Modeling for Computer Science	Python, R
9	Programming Languages **(LD, S)**	Java, Lisp, Prolog
10	Scripting Languages and Their Applications	Python, R
11	Parallel Architecture	C
12	Software Engineering	Varies
13	Web Programming	Javascript, HTML, CSS
14	Introduction to Artificial Intelligence	Varies
15	Computer Graphics	Varies

LD = Live Demo, S = Survey

With each new wave of users, we sought and received user feedback. Students provided feedback in-person, on class forums (e.g., Piazza), indirectly to teaching assistants and professors, and via email. Based on this feedback, we found and fixed bugs, and added new features.

To date, more than 3,000 students have signed up to use KODETHON in at least 15 different courses at UC Davis. Table 1 lists the courses where students have reported using KODETHON, where we have given live demonstrations, and where we distributed our user survey (§4). We gave live demonstrations in lower division programming courses, and upper division "Programming Languages" – where students are expected to program in Java, Lisp, and Prolog in a ten-week quarter. Student also reported using KODETHON in courses that we never gave demos too, *e.g.*, "Software Engineering",which indicates that KODETHON has been useful across our CS curriculum. Students have used KODETHON to write over 15.1 million lines of code. We do not count all of the intermediate code students wrote and delete/overwrote. The top-5 languages were C++ (4.5M), C (2.5M), JavaScript (2.2M), Java (1.4M), and C/C++ Header (1.2M). `C++` and `C` are used in large classes of about 300 students where students build projects that are hundreds of lines.

4 USER SURVEY

We developed a questionnaire consisting of 40 closed and open-ended questions, and scale items. Participants took 10 minutes, on average, to answer it completely. We asked about demographics, education, programming background [28], and general KODETHON usage. We also asked students about their perceptions of KODETHON using the measures defined in the USE Questionnaire [21]. Each items was scored on a 5-point Likert scale ranging from Strongly

Table 2: "I use Kodethon to." Multiple choices were allowed.

Activity	Participants ↓	Percentage of Participants ↓
Submit Assignments	117	84%
Test Programs	63	45%
Write Programs	46	33%
Collaborate	36	26%
Share Programs	18	13%
Other	1	1%

Disagree to Strongly Agree. We asked students perceptions about the impact of Kodethon in satisfying their learning objectives and their future career (Table 4). To learn about students' digital habits and their potential relationship with adoption of Kodethon, we asked students two questions: (1) "How often do you use Google Docs or Office 365[1]?" , and (2) "How often do you use stand-alone editors and IDEs such as Sublime, Atom, and Eclipse?"

We recruited participants by posting in Piazza class forums for two ongoing courses, and by emailing students from a third course from the previous academic quarter (Fall 2017) (see Table 1). We offered participants an entry in a drawing for ten $20 Amazon gift cards. Based on size of the classes, we estimate that we reached out to approximately 850 students in total from which 140 students chose to participate in our survey. The demographics of our participants were:

- Gender: Male (87), Female (51), Not identified (2).
- Ethnicity: Asian (90), White (35), Other (17), American Indian or Alaska Native (2), African American (1), Native Hawaiian or Pacific Islander (1).
- College: Letters and Science (91), Engineering (38), Agricultural and Environmental Sciences (8), Other (3).
- University Standing: Freshman (39), Sophomore (40), Junior (38), Senior (23).
- Years of Programming Experience: 0-1 (32), 1-2 (48), 2-3 (27), 3-4 (16), 4-8 (15), 8 or more (2).

5 RESULTS

5.1 Usage

Almost half of survey participants, 48%, reported using Kodethon often: Very Often (26, 19%), Often (41, 29%), Sometimes (23, 16%), Rarely (50, 36%), and Never (0, 0%). Table 2 shows that a large majority of participants, 83.6%, use Kodethon to submit assignments. It also shows that about a third of participants, 33%, use Kodethon to write their programs. This is consistent with our expectation that although not all students will opt to use Kodethon, a significant proportion will elect to use it. This is also consistent with our direct observations from previous courses where we observed about a third of all students opting to use Kodethon to complete the programming assignments. Most students tend to use the primary editor or IDE recommended by the instructor, for example, CLion or Eclipse.

Participants reported using Kodethon on a variety of devices. A great majority (95%) reported using Kodethon on their personal laptops. Interestingly, 8% use Kodethon on campus desktops. Indeed, we have directly observed students using Kodethon on campus desktops provided by the CS department which is interesting because those desktops already have ready-to-use programming environments. We hypothesize students find it more convenient to store their files on Kodethon where they can access their files from other devices. A large portion, almost half, of students used Kodethon in Windows. Students reported negligible use on other devices, *i.e.*, smart phones, tables, and other machines.

[1] Google applications and Office 356 are freely available to students.

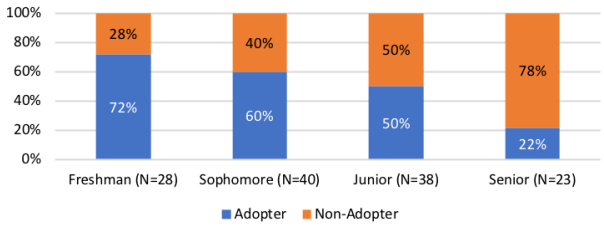

Figure 2: Adoption by students by university standing.

Table 3: Responses to the items in the Usefulness measure.

#	Item	SA	A	N	D	SD	Count
1	It helps me be more effective.	5%	27%	33%	20%	15%	134
2	It helps me be more productive.	5%	23%	30%	26%	16%	135
3	It is useful.	5%	50%	27%	8%	10%	136
4	It gives me more control over the activities in my life.	5%	16%	38%	22%	19%	129
5	It makes the things I want to accomplish easier to get done.	4%	21%	32%	26%	23%	135
6	It saves me time when I use it.	4%	24%	25%	29%	18%	138
7	It meets my needs.	4%	33%	32%	18%	13%	136
8	It does everything I would expect it to do.	6%	24%	25%	29%	17%	139
	Total	5%	27%	30%	22%	16%	1082

SA = Strongly Agree, A = Agree, N = Neither Agree nor Disagree, D = Disagree, SD = Strongly Disagree

5.2 Characteristics of Adopters

We classified students into two groups: 67 *adopters* and 73 *non-adopters*. Adopters are those who reported using Kodethon "Often" or "Very Often". The rest are non-adopters. We searched for individual factors that might correlate with adoption of Kodethon. We used Chi-square test for independence between adoption and 1) gender (p-*value* > 0.05), 2) university standing (p-*value* < 0.002), 3) programming experience (p-*value* > 0.05), and 4) university college (p-*value* > 0.05). We found that adoption correlates with university standing (p-*value* < 0.002). Students were less likely to adopt Kodethon as standing increased. We show the results in Figure 2. For digital habits, test of independence failed to find difference between the digital habits of adopters and non-adopters in using online editing tools such as Google Doc or other web IDEs ($p > 0.05$). However, Chi-square test suggested a difference in usage of stand-alone IDEs and editors between adopters and non-adopters ($p < 0.002$) which indicates that as students adopt Kodethon they tend to not use the alternative stand-alone tools and vice versa.

Table 4: Responses to additional Usefulness items.

#	Item	SA	A	N	D	SD	Count
1	It helps me become a better programmer.	4%	16%	43%	22%	14%	137
2	It is useful for real-world programming.	3%	29%	36%	19%	13%	136
3	It helps me develop skills I will need in software engineer jobs.	4%	21%	41%	22%	10%	135
4	It helps me focus on the important aspects of programming.	5%	20%	38%	24%	12%	138
	Total	4%	22%	40%	22%	12%	546

SA = Strongly Agree, A = Agree, N = Neither Agree nor Disagree, D = Disagree, SD = Strongly Disagree

5.3 Perceptions

To measure this attitude, we used the Usefulness measure defined by Lund in the USE questionnaire [21]. We summarize the responses in Table 3. The results of the survey are mixed. Considering all items, out of 1, 082 responses, 32% (348) agreed on the usefulness of Kodethon, 30% (325) were neutral, while 38% (409) disagreed. One of the most interesting items was the direct statement, "It is useful."; 55% of participants agreed. We conjecture that participants find different aspects of Kodethon useful and they converge under this broad statement. Another interesting item is "It does everything I would expect it to do." to which only 30% of participants agreed. This tells us that although we built many features into Kodethon, students still expect more from a web IDE. In future work, we plan to investigate what else students expect Kodethon to do.

We developed additional usefulness items to get a sense of students perceptions about Kodethon's usefulness beyond the classroom. Table 4 summarizes the responses. The results of this section were a bit shocking to us, as we were expecting for more positive feedback. The most surprising agreement rate was for "It helps me focus on the important aspects of programming" that only 25% of students agreed. We expected higher agreement given that Kodethon allows students to simply write and run code without having to install anything. It shows that there is a gap between our intended impact of the IDE, as its developers, and the students' perceptions. One explanation can be that the Kodethon has failed to adequately match students' programming needs. An alternative explanation is that students may have varying opinions on what is "important" or even what is "programming." We were surprised that only about a third of participants, 32%, feel that Kodethon can be useful for "real-world programming." However, it is in contrast to observations in [4] that a vast majority of students perceived programming in a web IDE as real programming. We should note that it is unclear to us to what "real-world programming" actually means to students; regardless of the intended meaning and its accuracy, majority of students perceived that Kodethon does not prepare them for real programming.

Lastly, for all items, a significant fraction of responses were neutral. We expected most students to agree or disagree. One possible explanation is that, as many participants reported, they do not use Kodethon often enough to form an opinion. Another possible explanation is that students may feel that they have not been exposed to enough IDEs to form an opinion. We did not find that any participant simply replied the same to all items. It would be interesting to investigate this deeper and explore if there are missing features or usability traits that would cause participants to shift from neutral to agreement that Kodethon is useful.

Table 5: The top-10 responses to "What features of Kodethon do you find most useful?"

Feature	Participants ↓	Percentage of Participants ↓
Web-based	90	65%
No Installation Required	85	61%
Assignment Grading	73	52%
Works on Multiple Devices	52	37%
Unix Terminal	49	35%
Assignment Feedback	48	35%
Real-time Collaboration	46	33%
Programming Language Support	35	25%
File Cloud Storage	32	23%
Syntax Highlighting Code Editor	32	23%

Table 6: The top-10 categories of open-ended responses to "List the most positive aspect(s) of Kodethon."

Feature	Responses ↓	Percentage of Responses ↓
Assignment Grading and Feedback	55	21%
Real-time Collaboration and Chat	54	21%
Easy to Use	36	14%
Web-based	18	7%
File Cloud Storage	18	7%
Other	16	6%
Unix Terminal & CDE Shell	13	5%
Programming Language Support	13	5%
No Installation Required	3	9%
Smart Run Button	3	7%
Total	262	100%

We provided participants with a list of 18 features, and asked them to select which features they found most useful. Table 5 shows a summary of the responses. The most important finding is that what participants find useful corresponds with our hypothesis that many students would find a web IDE useful because it is a convenient, low-threshold, ready-to- use programming environment. Participants selected "Web-based" (65%) and "No Installation Required" (61%) as the top two features.

We asked participants to list the most positive aspects (up to 3) in their own words. We coded 262 responses into 16 categories, shown in Table 6. Participants mentioned many of the same features we provided earlier but in their words. However, here students mention the LMS (automatic grading and feedback) 55 times, and the real-time collaboration features 54 times. Participants also commented on the usability of Kodethon saying it was easy to use 36 times and easy to learn 2 times. Here are some example responses:

> "I appreciate the cloud storage a lot. It saved my grade when my laptop broke."
>
> "I like how you don't need to install anything, for beginners this is helpful."

We also asked participants to list the most negative aspects (up to 3) in their own words. We coded 237 responses into 14 categories. 56 participants reported difficulties or issues or difficulties with the user interface. 37 participants mentioned experiencing issues with file saving. 34 participants responded simply "buggy" or "glitchy" without elaborating on which feature. A large fraction of the issues experienced by users have been due to heavy load on the system.

As a result, we responded by increasing the nodes in our cluster, and making performance improvements.

6 BROADER LESSONS

We discuss some broader lessons resulting from interactions with students and instructors and supported by our student adoption and perception findings.

A web IDE is not a silver bullet: As our results show, a web IDE is not an ideal tool for every student. First, many students do not struggle with or do not mind installing programming tools. Second, web IDEs introduce different challenges. For example, the user interface is different and can take effort and time to learn. While many users found Kodethon easy to use, many others felt otherwise. Another example, because Kodethon is a web application, fluctuations in network speed can cause the user to experience latency or even downtime. Even if a user enjoys the user interface and performance, network glitches can negatively impact their experience; a non-issue in desktop IDEs.

Cost of Building a web IDE: Based on our experience, we recommend caution before selecting a web IDE and especially before deciding to build a custom one. This advice should be applicable to educators and researchers considering the question of introducing a web IDE in their curriculum. At the time we began developing Kodethon, there were few existing web IDEs, namely Runnable[26], Koding [18], Nitrous [20], and Cloud9 [14]. None of them were designed for students and none of them were customizable to the needs at UC Davis. So we decided to build our own. We learned that building a web IDE that is reliable and scalable for *real use* is an expensive endeavor with a lot of technical challenges. With two engineers, it still took almost four years to reach our current state. Nonetheless, it was a good decision considering that Runnable and Koding have since changed focus, and Nitrous has shut down. While others have surfaced like CodeAnyWhere [13] and CodeEnvy [5], we have been able to leverage our architecture to introduce a learning management system to help instructors grade assignments automatically and provide students with instant feedback, a feature which 52% of participants found useful.

Some students prefer a web IDE: Usually, instructors tend to select and present a single IDE, *e.g.*, CLion. Instead, instructors can provide students with multiple programming environment options, including a desktop and a web IDE, and encourage students to choose one, weighing the benefits and trade-offs. This is consistent with the principle that every student learns differently and with previous work [12]. And as our results show, many students will voluntary opt to use a web IDE and adopters tend to not use standalone IDEs. Moreover, students adopted Kodethon not merely because it was novelty but because it provided tangible benefits to them, ease of use, convenience, and portability. As new generation of students uses the web more often and longer [22, 29], a web IDE can be a more familiar system for them to interact with than traditional IDE. Moreover, cloud storage of files seems to assure students that their programs and homework will not be lost due unexpected hardware or software failure on their machines. Kodethon also provides them with an easy way of collaborating in real-time to pair program without having to resort to tools like `git` which often introduce more complexities [6].

Learning objectives need to be explicit and clear: In discussions with instructors, we have learned that they often have hidden learning objectives, often unknowingly. One is to learn how to install and configure development tools. The rationale is that there is value in such a skill. A problem is that this learning objective is rarely made explicit or evaluated. On balance, some instructors have found that it is more important to give students a ready-to-use programming environment so they can spend more time learning to actually write code. Another problem with this hidden objective that it is too vague; it does not specify which tools (IDEs, text editors, compilers/interpreters, *etc.*) should be learned.

Students may confuse programming and system building: Based on conversations with students, many do not consider programming in a web IDE to be "real programming". This negative view may stem from the fact that some students confuse computational thinking and programming with system building. In system building, developers must consider the production environment and compile, configure and tests their programs to meet some requirements, while computational thinking concerns with creating an appropriate abstraction of the problem domain and encoding the abstraction in a given programming language. A large portion of topics in computer science curricula, especially in the introductory courses, focus on the latter. Moreover, system building is still possible in a web IDE like Kodethon. In fact, there are a growing number of professional programmers working primarily on web IDEs, like Amazon Cloud9 [14], developing on cloud instances.

7 THREATS TO VALIDITY

The data presented in the paper is based on a quantitative anonymous survey study. The results are limited to responses to the questions. Students background can impact the generalization of the results beyond this study. We deployed the system in a public, selective university, where most students have some prior programming experience. Prior experience can influence their view of educational programming tools. For example, they may already know how to set up and configure the programming tools; in that case, a web IDE is of little value to them. However, in our survey, majority of students, even some non-adopters, indicated that a web IDE is a useful tool.

8 CONCLUSION

We described a web IDE and its deployment in a large public university that has been used by more than 3,000 students in multiple programming courses. We used a quantitative survey study to evaluate students satisfaction and perception of the system. We found that students in the introductory classes tend to adopt a web IDE in early programming classes than more advanced ones. We found no strong correlation between adoption of online editing tools (e.g., Google Doc) and the web IDE. The analysis of results suggests mixed reaction to the adoption and use of the IDE. While around a third of students agree on the usefulness of the IDE in their classes, another third disagree. The results suggests that a web IDE is not silver bullet; it is not an ideal educational tool for every student under any circumstances. We also caution about the cost of developing custom-made web IDE systems.

REFERENCES

[1] Khan Academy. 2017. *Khan Academy.* https://www.khanacademy.org/

[2] Amjad Altadmri, Neil C.C. Brown, and Michael Kölling. 2015. Using BlueJ to Code Java on the Raspberry Pi. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 178–178. https://doi.org/10.1145/2676723.2691872

[3] Valerie Barr and Deborah Trytten. 2016. Using turing's craft codelab to support CS1 students as they learn to program. *ACM Inroads* 7, 2 (2016), 67–75.

[4] Luciana Benotti, Federico Aloi, Franco Bulgarelli, and Marcos J. Gomez. 2018. The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 2–7. https://doi.org/10.1145/3159450.3159579

[5] CodeEnvy. 2019. *CodeEnvy.* https://codenvy.com/

[6] Santiago Perez De Rosso and Daniel Jackson. 2016. Purposes, Concepts, Misfits, and a Redesign of Git. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016)*. ACM, New York, NY, USA, 292–310. https://doi.org/10.1145/2983990.2984018

[7] Gregory Dyke. 2011. Which Aspects of Novice Programmers' Usage of an IDE Predict Learning Outcomes. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education (SIGCSE '11)*. ACM, New York, NY, USA, 505–510. https://doi.org/10.1145/1953163.1953309

[8] Kasper Fisker, Davin McCall, Michael Kölling, and Bruce Quig. 2008. Group Work Support for the BlueJ IDE. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '08)*. ACM, New York, NY, USA, 163–168. https://doi.org/10.1145/1384271.1384316

[9] Corey Ford and Clinton Staley. 2016. Automated Analysis of Student Programmer Coding Behavior Patterns (Abstract Only). In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*. ACM, New York, NY, USA, 688–688. https://doi.org/10.1145/2839509.2850540

[10] Max Goldman, Greg Little, and Robert C. Miller. 2011. Real-time Collaborative Coding in a Web IDE. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 155–164. https://doi.org/10.1145/2047196.2047215

[11] Philip J. Guo. 2013. Online Python Tutor: Embeddable Web-based Program Visualization for Cs Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 579–584. https://doi.org/10.1145/2445196.2445368

[12] Juha Helminen, Petri Ihantola, and Ville Karavirta. 2013. Recording and Analyzing In-browser Programming Sessions. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/2526968.2526970

[13] Codeanywhere Inc. 2019. *CodeAnywhere.* https://codeanywhere.com/

[14] Cloud9 IDE Inc. 2019. *Cloud9 IDE.* https://c9.io/

[15] Jam Jenkins, Evelyn Brannock, and Sonal Dekhane. 2010. JavaWIDE: Innovation in an Online IDE: Tutorial Presentation. *J. Comput. Sci. Coll.* 25, 5 (May 2010), 102–104. http://dl.acm.org/citation.cfm?id=1747137.1747155

[16] Ayaan M. Kazerouni, Stephen H. Edwards, T. Simin Hall, and Clifford A. Shaffer. 2017. DevEventTracker: Tracking Development Events to Assess Incremental Development and Procrastination. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '17)*. ACM, New York, NY, USA, 104–109. https://doi.org/10.1145/3059009.3059050

[17] Caitlin Kelleher and Randy Pausch. 2005. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.* 37, 2 (June 2005), 83–137. https://doi.org/10.1145/1089733.1089734

[18] Koding. 2017. *Koding.* http://www.koding.com/

[19] Michael Kolling, Bruce Quig, Andrew Patterson, and John Rosenberg. 2003. The BlueJ System and its Pedagogy. *Computer Science Education* 13, 4 (2003), 249–268. https://doi.org/10.1076/csed.13.4.249.17496

[20] Frederic Lardinois. 2016. *Cloud development platform Nitrous.io shuts down.* https://techcrunch.com/2016/10/31/cloud-development-platform-nitrous-io-shuts-down/

[21] Arnold Lund. 2001. Measuring Usability with the USE Questionnaire. *Usability Interface* 8, 2 (2001), 3–6.

[22] Sidneyeve Matrix. 2014. The Netflix effect: Teens, binge watching, and on-demand digital media trends. *Jeunesse: Young People, Texts, Cultures* 6, 1 (2014), 119–138.

[23] Jonathan P. Munson. 2017. Metrics for Timely Assessment of Novice Programmers. *J. Comput. Sci. Coll.* 32, 3 (Jan. 2017), 136–148. http://dl.acm.org/citation.cfm?id=3015220.3015256

[24] James H Paterson and John Haddow. 2007. Tool Support for Implementation of Object-Oriented Class Relationships and Patterns. *Innovation in Teaching and Learning in Information and Computer Sciences* 6, 4 (2007), 108–124. https://doi.org/10.11120/ital.2007.06040108

[25] James H. Paterson, John Haddow, Miriam Birch, and Alex Monaghan. 2005. Using the BlueJ IDE in a Data Structures Course. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '05)*. ACM, New York, NY, USA, 349–349. https://doi.org/10.1145/1067445.1067548

[26] Runnable. 2017. *Runnable.* http://www.runnable.com/

[27] Carsten Schulte and Jens Bennedsen. 2006. What Do Teachers Teach in Introductory Programming?. In *Proceedings of the Second International Workshop on Computing Education Research (ICER '06)*. ACM, New York, NY, USA, 17–28. https://doi.org/10.1145/1151588.1151593

[28] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2014. Measuring and modeling programming experience. *Empirical Software Engineering* 19, 5 (01 Oct 2014), 1299–1334. https://doi.org/10.1007/s10664-013-9286-4

[29] Victor C Strasburger, Marjorie J Hogan, Deborah Ann Mulligan, Nusheen Ameenuddin, Dimitri A Christakis, Corinn Cross, Daniel B Fagbuyi, David L Hill, Alanna Estin Levine, Claire McCarthy, et al. 2013. Children, adolescents, and the media. *Pediatrics* 132, 5 (2013), 958–961.

[30] Arie van Deursen, Ali Mesbah, Bas Cornelissen, Andy Zaidman, Martin Pinzger, and Anja Guzzi. 2010. Adinda: A Knowledgeable, Browser-based IDE. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2 (ICSE '10)*. ACM, New York, NY, USA, 203–206. https://doi.org/10.1145/1810295.1810330

[31] Ke Wang, Benjamin Lin, Bjorn Rettig, Paul Pardi, and Rishabh Singh. 2017. Data-Driven Feedback Generator for Online Programing Courses. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, USA, 257–260. https://doi.org/10.1145/3051457.3053999

[32] Ke Wang, Rishabh Singh, and Zhendong Su. 2018. Search, Align, and Repair: Data-Driven Feedback Generation for Introductory Programming Exercises. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, USA.

[33] Lisa Wang, Angela Sy, Larry Liu, and Chris Piech. 2017. Deep Knowledge Tracing On Programming Exercises. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale - L@S '17*. ACM Press, New York, New York, USA, 201–204. https://doi.org/10.1145/3051457.3053985

[34] Michael Whitney, Heather Lipford-Richter, Bill Chu, and Jun Zhu. 2015. Embedding Secure Coding Instruction into the IDE: A Field Study in an Advanced CS Course. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 60–65. https://doi.org/10.1145/2676723.2677280

[35] Jun Zhu, Heather Richter Lipford, and Bill Chu. 2013. Interactive Support for Secure Programming Education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 687–692. https://doi.org/10.1145/2445196.2445396