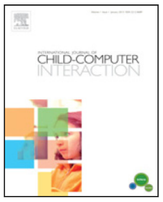
International Journal of Child-Computer Interaction xxx (xxxx) xxx

Contents lists available at ScienceDirect

International Journal of Child-Computer Interaction

journal homepage: www.elsevier.com/locate/ijcci



The cloud is the limit: A case study of programming on the web, with the web

David Weintrop [a,*], David Bau [b], Uri Wilensky [c]

[a] College of Education, College of Information Studies, University of Maryland, United States
[b] Computer Science, Massachusetts Institute of Technology, United States
[c] Center for Connected Learning and Computer-based Modeling, Northwestern University, United States

ARTICLE INFO

Article history:
Received 8 December 2017
Received in revised form 19 November 2018
Accepted 15 January 2019
Available online xxxx

ABSTRACT

The last ten years have seen a proliferation of introductory programming environments for younger learners. Increasingly, these environments are moving into the "cloud" where they can be accessed through web browsers and run on a variety of devices including tablets and smartphones. The shift to online settings enables a variety of powerful pedagogical features to be incorporated into the design of these learning environments, including making it easy to share learner-authored programs, browse projects written by others, and allow learners to incorporate various Internet resources into their work. Further, the Internet itself can serve as a productive canvas upon which novice programmers can create in the form of dynamic and interactive web pages. This shift in venue for authoring and editing programs is particularly well-suited for young learners growing up in an increasingly online world. In this paper, we present theoretical and practical arguments for online introductory programming environments as powerful learning tools and present data showing various ways young learners take advantage of features of the environment enabled by being situated online. In particular, the paper looks at how the online context can support young learners in authoring programs and interacting with programs authored by others. The contribution of this work is to advance our understanding of how the Internet can be utilized as a resource to situate learning and serve as an inviting and accessible pathway into computing.

1. Introduction

Today's learners live in an online world. From coordinating assignments through social media and participating in online courses to turning in homework assignments via Google Classroom and watching lessons in flipped classrooms on YouTube, the Internet is becoming an increasingly common context for learning. The ubiquity of the Internet in and out of school corresponds with, and contributes to, the growing importance of young learners being able to fully participate in computational contexts through the development of essential computational literacy skills. These skills and practices, recently collected under the umbrella term "Computational Thinking", have far-reaching applications across a diverse set of content areas and domains [1–4]. Central to our conceptualization of computational thinking is the ability to encode ideas in a way that a computer can execute, along with being able to interpret instructions written by others, creating what diSessa [1] calls a "two-way literacy". In our increasingly online and connected world, the Internet can serve as an accessible, familiar, and powerful learning context in which to develop these computational skills.

Situating introductory programming learning experiences online provides a compelling context for young learners to meaningfully engage in authentic programming practices that carry social and cultural meaning. Features of these environments include making it easy for learners to share programs and collaborate on projects, providing low barriers to entry, and the inclusion of a diverse array of scaffolds and designs to engage and support novice programmers. Having the learning environment online also allows young learners to pull in resources from the Internet into their projects, blurring the boundaries of the learning environment and the larger web. Thus, a growing number of introductory programming tools are being developed to run in a web browser and leverage various affordances of situating computational learning online.

This paper seeks to lay the theoretical groundwork for why situating programming experiences online can be productive for young learners and present data showing how design features supported by the online context contribute to creating an engaging and inviting introduction to computing. In particular, the paper looks at how design features enabled by the online context can support meaningful and productive interactions for children learning to program. This paper contributes to our understanding of online learning of computational ideas, and more broadly, the power of aligning learning experiences with technologies that hold social

* Corresponding author.
E-mail addresses: weintrop@umd.edu (D. Weintrop), davidbau@mit.edu (D. Bau), uri@northwestern.edu (U. Wilensky).

https://doi.org/10.1016/j.ijcci.2019.01.001
2212-8689/© 2019 Elsevier B.V. All rights reserved.

and cultural meaning, especially as they relate to youth growing up in a connected, online world. To explore these ideas, this paper presents a case study of Pencil Code, an online programming environment that incorporates contemporary Internet technologies to make it easy for learners to author and share their own interactive websites. The paper details various ways the design of Pencil Code incorporates Internet technologies and allows learners to incorporate resources from the larger Internet into their projects. Next, using data from a pair of classroom studies, we show how students used the various Internet-supported features of Pencil Code to have authentic and meaningful programming experiences. The paper concludes with a discussion of the ways that programming in the cloud provides an effective and accessible way to prepare learners for the digital futures that await them.

2. Theoretical framework & prior work

2.1. Theoretical framework

From the beginning, the idea that computers could serve as powerful contexts for learning was rooted in constructivist learning theory [2]. Papert, in creating the Logo programming language, sought to put the power of computing in the hands of young learners by designing a language that was accessible and supported expression, encouraged exploration, and empowered the learner to pursue projects they found personally meaningful. Additionally, Logo and constructionist learning activities incorporated the creation of public artifacts that could be presented, shared, and serve as a topic of conversation and object for reflection. These foundational design features have reverberated through the world of computer-based learning environments and are featured prominently in many of the more successful novice programming tools [5]. Papert's Logo emphasized powerful ideas and practices over memorized facts, an objective shared by many of the increasingly diverse tools that are being used to introduce learners to programming and computational thinking more broadly [6,7]. Where meaningful engagement and expressive programs with Logo involved drawing geometric landscapes, playing with sentence structures, and writing software for peers, the modern incarnation of Logo's design principles are increasingly grounded in the ability to write and share programs online [8,9]. In doing so, novices are introduced to programming by creating games, models, and stories that can be seen and played by others around the world, and participating in a global community connected through websites, forums, and other social spaces [10]. This form of online participation is characteristic of the Web 2.0 culture, where users contribute to the content on the web, acting as producers along with consumer [11].

An important component of Papert's constructionist design philosophy was the creation of "low-threshold, high ceiling" learning environments. "Low-threshold" speaks to the ease-of-entry for novices - that care should be taken to support learners in having early successes and design to enable various forms of engagement. The companion to "low-threshold" is "high-ceiling", meaning introductory tools should also be powerful computing environments, where users are not limited by the capabilities of the tool, and as experience and sophistication grow, the tool can continually support the ideas of the learner. This can take a number of forms, including designing an environment that is powerful enough to be used by experts or providing a seamless transition from early learning experiences with the technology to a fully featured expert version of the tool. As we will demonstrate below, bringing novice programming environments online provides multiple opportunities to design for both "low-threshold" and "high-ceiling" engagement.

A critical dimension of a "high-ceiling" learning environment is how it can impart a sense of authenticity to the learner. Shaffer and Resnick [12], in their review of authenticity in education, identify four types of authentic education: (1) activities alignment with outside world, (2) content aligned with what learners want to know, (3) methods of inquiry aligned with a discipline, and (4) assessment aligned with instruction. For the first three of these forms of authenticity, online programming environments align well with authentic education. As new interfaces and environments are developed for introductory programming, consideration of authenticity is important, especially with older learners, as research has found that high school aged learners are concerned with the real-world applicability of the programming language being learned [13,14]. By introducing learners to programming on the web, and enabling them to develop websites using the same technologies that professionals use, the learning context aligns with authentic learning and parallels professional programming practices. Engaging in recognizable practices of the field and producing artifacts that are similar to what experts and professionals produce, helps legitimize the learning activity and enable the learner to identify as a practicing member of that community [15].

2.2. Introductory programming environments and the web

Beginning with Logo in the 1970s, a long line of programming languages, environments, and activities have been designed to make programming accessible and intuitive for learners (for a review or novice programming environments, see: [5–7]). In the last 15 years, these environments have increasingly incorporated online components to engage and educate novices. The focus of this paper is to explore potential benefits and drawbacks associated with this shift.

When we use the term *Online Programming* we are referring specifically to the act of authoring programs using source code editors whose contents are served via a web server and are rendered and used inside a web browser. This context for writing programs is in contrast to standalone, offline programming software ranging from light-weight source code editors (e.g. Sublime Text, Emacs) to fully-featured integrated development environments (e.g. Eclipse, NetBeans). This conceptualization of online programming is intended to take literally the subtitle of the article: programming on the web, with the web. In these online programming environments, the authoring of programs is embedded within and supported by the infrastructure of the Internet directly. While it is important to note that many standalone editors have features that connect with the Internet, such as the ability to collaborate with others within the editor or access online materials from within the editors itself, this is different than the fully-online form of programming discussed in this paper. That being said, the benefits discussed below are not exclusive to the type of online programming environment that is the focus of the paper. For example, offline code editors can still cultivate active online communities and leverage social networks as a means to improve the programming experience. This paper is not arguing that only online programming environments can leverage features of the Internet to promote engagement and learning, but rather that such learning and engagement happens especially felicitously with online programming tools.

2.2.1. Programming on the web

In the last few years, as online educational platforms mature and diversify, an increasing number of tools are adopting a programming-in-the-browser approach by developing and embedding code editors in websites alongside the web pages that host the content. Platforms like Khan Academy, Codecademy, and Code.org's Code Studio all use in-browser development tools. In some cases, these in-browser learning environments provide a fully featured integrated development environment that has many of the features of professional tools. Likewise, stand-alone environments designed to emphasize creativity and engagement, independent of specific curricula, have also moved online. Environments like Scratch, NetLogo, and ToonTalk that were initially designed to run locally on the learner's computer have been rewritten to run in the browser [16–18]. Notable in this effort is the Blockly library which is an HTML and JavaScript library designed to make it easy to integrate graphical, block-based programming into existing website and projects [19]. There are a number of benefits to having the entire programming environment reside on the cloud and be accessed through a web browser, including portability, ease of sharing and accessing work, and the previously mentioned pedagogical and authenticity dimensions. We discuss these and other benefits in more detail later in the paper.

2.2.2. *Programming with the web*

While the online environments discussed in the previous section are hosted online, learners are not necessarily writing programs in the language of the Internet, instead, written programs can only be run in the context in which they are written. For example, a program written as part of a Codecademy lesson can only be run inside Codecademy — this constrains the program, how it can be accessed, and what can be created with it. Thus, despite the freedom of being online, learners are still constrained to only view and run their programs through the environments in which they were written. In contrast, other learning environments on the Internet are designed to make the web itself the canvas for learning. For example, Mozilla's X-ray Goggles tool (http://webmaker.org/goggles) gives learners the ability to see how live websites are constructed. Using this tool, learners can load any web page, and with the click of a button, begin to edit content, change visual features or alter the dynamic behaviors of the site. Here, the coding still lives in the browser, but now the Internet is acting as both the programming environment and the canvas. A benefit of making web-based content the focus of introductory learning is the transparency of the Internet as a platform. On any web page, a single click can give the user a peek "under-the-hood" to see how the page was created. Further, the plain-text formats of HTML and JavaScript make it easy to parse and reuse for other projects — enabling Scratch-style remixing, but now with the whole Internet as the library of potential examples. Another example of this can be seen in the Snap! programming environment [20], which includes a `url` block that allows the user to pull the contents of a web page into their program. We see these features of programming on the web, with the web, as particularly compelling and will demonstrate how learners pull resources from the larger web into their projects, blurring the lines between the learning environment and the Internet.

2.2.3. *Social aspects of online programming*

A third dimension that online learning can bring to introductory environments is using the Internet itself as a platform for learners to interact with each other and the programs they have authored. An early programming environment that productively leveraged online, connected learning was MOOSE Crossing [21], which incorporated social aspects to programming by allowing learners to programmatically create worlds for others to explore and enable synchronous communication between users. Work in this environment shows how online contexts can provide productive peer-mentoring learning opportunities [21]. Along with the synchronous communication, a growing number of online environments include mechanisms for asynchronous communication in the form of message boards and project galleries. The Scratch programming environment and its accompanying website has been very successful in building an online community of learners [22]. Along with a number of language and interface features designed to make programming intuitive and engaging [23], the Scratch website allows learners to share, browse, and remix projects written by others. Through this forum, learners are able to engage in a variety of forms of participation in the Scratch community, including asking for advice from more experienced users, give feedback on both aesthetics and technical aspects of projects, and sharing advice from their own experiences working with Scratch [24]. Similar online support can be found for other introductory programming learning environments like Looking Glass Alice [25] and NetLogo [26].

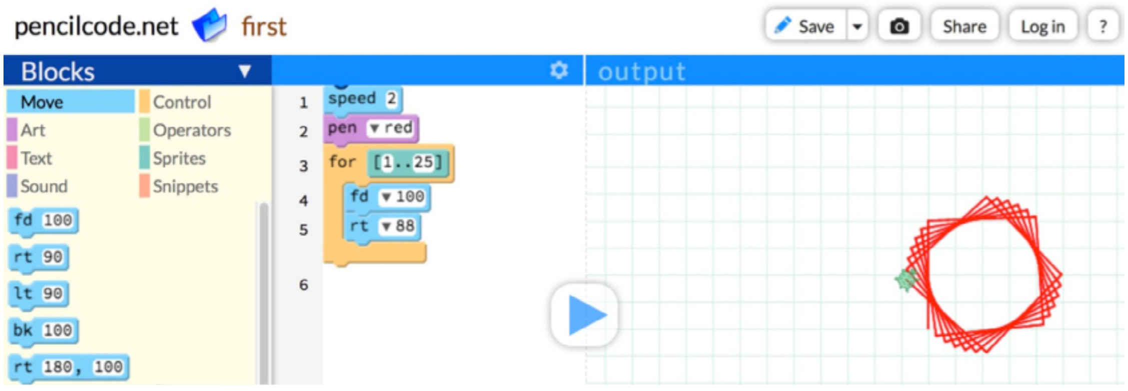
3. Meet pencil code

This study uses the Pencil Code environment to demonstrate the affordances of programming on the web, with the web for young learners. In this section, we introduce Pencil Code and highlight a few of its design features.

Pencil Code is a fully cloud-based, online environment for learning to program [27]. The editor runs in a browser, and students save, edit, share, and publish their work online. Its interface (Fig. 1) is split into two panes: on the left is a dual-modality programming editor that supports both visual block-based and conventional text-based code, while the right side is a web page that visually renders the program the learner is writing. Pencil Code embeds all student work in the Web: every student project is comprised of JavaScript and HTML so can be run in any modern browser and has a unique, accessible URL that can be used to share, view, and run the program. Student work is not obfuscated: using "view source" on a published web page shows the student code clearly inside simple HTML code.

Pencil Code was designed to encourage two main types of programming activities. In the spirit of the Logo language, traditional coding concepts such as loops, conditionals, and functions can be incorporated into turtle graphics drawing programs starting from a single line of code such as `fd 100`. This is accomplished with the Pencil Code turtle library, which acts as a supplement to the standard JavaScript used in the editor. Similarly, the visual block-based interface is a mode for editing code, not a separate language, a fact made clear by the editor's ability to move back-and-forth between block and text presentations of the code [28]. The second type of programming activities supported is creating live, interactive web pages with HTML images, buttons, animation and music, that will appear no different to a visitor to the page than any other website online. Pencil Code is a "high ceiling" learning environment that is careful to avoid placing artificial barriers around the learner. Students can edit JavaScript code of any level of sophistication with either blocks or text. Several widely used professional programming libraries such as jQuery are also included, so coding examples from general web programming tutorials can be used directly in Pencil Code programs without modification.

The environment also provides several easy pathways from the Internet into student programs and vice-versa: for example, a turtle can "wear" an image found on the Internet using the `wear` command. To pull in an existing web resource (like an image or video), the user need only copy and paste its URL into the program. Along with using full URLs for the `wear` command, the student can also type in descriptions of images and Pencil Code will search through the Creative Commons image repository to find a related image, so programming an object to `wear grumpy cat` will result in the turtle being replaced by an image of the sour-looking cat you expect. This ease of incorporation of online resources further lowers the barrier to expanding beyond the Pencil Code sandbox and shows that Pencil Code and the programs learners write are online resources like any other web page they might visit. Pencil Code also allows students to include HTML and CSS in their projects, the standard technologies used to define the content and formatting for web pages. Students can programmatically write



```
speed 2
pen red
for [1..25]
  fd 100
  rt 88
```

Fig. 1. Pencil Code's Interface with the coding area on the left and program output on the right.

content into their projects by including HTML in their program directly, or by defining a starting world for a program by directly editing the HTML and CSS context for their project in a scaffolded environment. Because Pencil Code has the jQuery library embedded within it, students can access the DOM elements they create and manipulate them using the same library as professional web developers. As with scripts, HTML can be edited in the left pane in a dual-modality editor. Pencil Code's use of standard Internet technologies across all aspects of the environment, including the languages learners use to create their projects, the projects they create, and the larger environment in which the programming is situated, makes clear that nothing is hidden from the learner, they are programming on the Internet, with the Internet.

4. Methods

In exploring the affordances of programming on the web, with the web, we use two classroom implementations of introductory computer science materials using Pencil Code. In both studies, the curricular materials were designed by the authors, with education researchers present in the classroom observing and providing technical support. A case-study approach is used in this work as a means to highlight the affordances of learning to program in online contexts and to document specific forms of interaction enabled by the online format. The assignments and events reported below are intended to serve as typical interactions that demonstrate how the online setting and forms of interactions supported by the environment had a positive impact on learners' experiences.

4.1. Context and participants

The first study took place over the first five weeks of an introduction to programming course at a selective enrollment public high school in a Midwestern city. Ninety students across three sections of the class spend roughly 45 min of class time a day working in Pencil Code. These classes were 83% male, racially diverse (41% White, 27% Hispanic, 11% Asian, 10% African American, 11% Multiracial), and included student from all four years of high school: 16% freshmen (ages 14–15), 41% sophomores (ages 15–16), 29% juniors (ages 16–17), 14% seniors (ages 17–18). Just under half of the students (47%) responded that they speak a language other than English at home. Fifty-nine percent of the students in the school are from low-income families. During the five-week Pencil Code unit, students worked on a number of projects intended to introduce them to various core programming concepts, including variables, functions, and conditional and iterative logic (the full curriculum can be found in Appendix A of [29]).

The second set of classroom data comes from a once-a-week 100-minute-long all-female colloquium course called Computational Thinking for Girls (CT4G) taught at a different public high school in the same Midwestern city. Over the course of three consecutive classes, the girls in the class were introduced to various features of Pencil Code as they worked towards building a website that would advertise the class they were participating in. Thirteen girls (2 8th graders (age 14), 8 freshmen (ages 14–15), 2 sophomores (ages 15 and 16), and 1 junior (age 16) participated in the class. The school population is 72% African American, 25% Hispanic, and less than 2% each white and Asian, a distribution that is reflected in the CT4G classes. Sixty-seven percent of students in the school are from low-income families. Further details about the setting and the CT4G program can be found in [30,31].

4.2. Procedures

Both of the implementations followed the same set of research procedures and data collection strategies. Every class session began with the instructor in the front of the classroom introducing the assignment and topics of the day. Every student sat in front of their own computer so could author their own programs. The details of the assignments used in the case studies are described in greater detail below. The researchers created accounts for each of the students and a portal for them to use to easily access their Pencil Code accounts. The programming environment students used was instrumented so every time a student ran a program, the contents of the program, along with program metadata (the author, the assignment, a timestamp, etc.) were collected on a remote server. This gave the researchers access to all of the projects that the students authored.

During the class sessions, field notes were taken by the present researchers, paying particular attention to social interactions that emerged between learners as well as unique or interesting programs written by learners that could be further analyzed by the research team outside of class time. This data source is a useful supplement to the log data collected as it provides further context for how features of the online programming environment supported positive interactions, particularly with respect to the social aspects of children learning to program.

5. Findings

5.1. From Bieber to basketball to battleship

At the conclusion of both classroom studies, students were given an open-ended summative assignment that asked them to create a final project that incorporated the concepts covered in class (loops, variables, etc.). Given that Pencil Code is designed to allow learners to program on the web, with the web, many students pulled in resources from the larger Internet as a means to create projects that reflected their interests. Across the 95 final

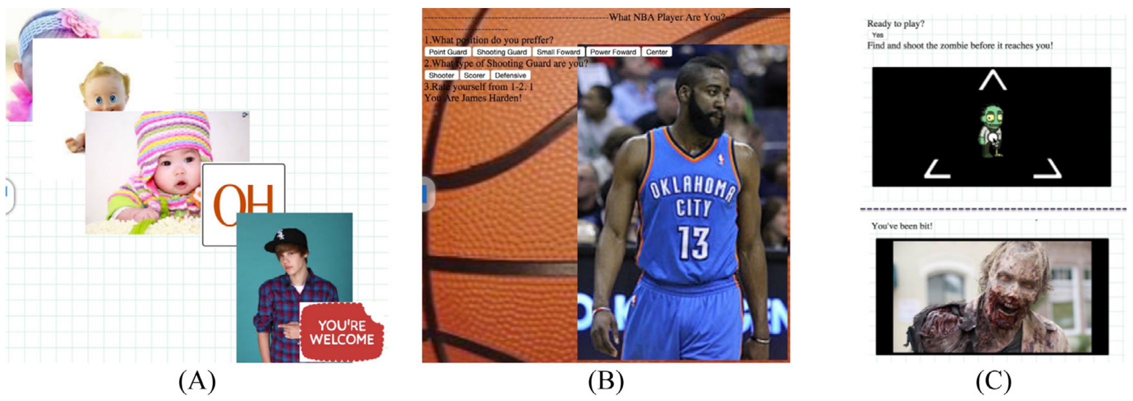


Fig. 2. Three examples of media-rich projects that incorporated resources for the Internet.

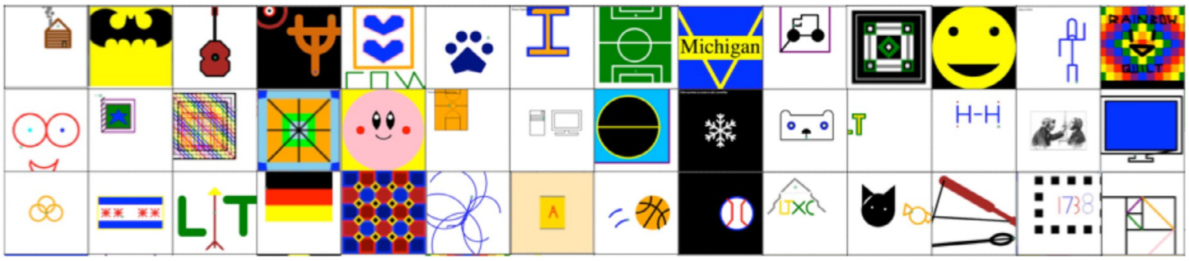


Fig. 3. A portion of the class quilt web page comprised of patches written by students.

projects submitted in the two settings, online resources shaped the assignment both in terms of the content of the projects themselves and in how students shared and discussed their work. For example, one student in the CT4G class decided to make her final project a recreation of her favorite Justin Bieber Song, "Baby", by writing a program to animate the chorus of the song, which used iterative logic to move through various pictures of babies pulled from the Internet and culminated in a picture of Justin Bieber and a sign saying "You're Welcome" (Fig. 2A). When the class was given a chance to share their work at the end of the last day of the Pencil Code unit, this student eagerly volunteered to present her project, getting the class to join in and sing along with her and her program.

Along with musicians, student projects included movie stars, superheroes, and athletes, as well as images of organizations students, were affiliated with and pictures of their friends pulled in from social media sites. It was not just that students created websites with static sets of images for others to view, students also used the Pencil Code platform as an opportunity to show off and share their knowledge of other topics. For example, one student who is a big basketball fan created a website that included an interactive questionnaire about basketball, asking users about their preferences in terms of what their favorite position was and what style of play they liked. Based on the responses given, the program would tell the user who their favorite basketball player should be, including information and images about that player (Fig. 2B). Other students followed a similar template, writing surveys to recommend TV shows and to quiz users about their knowledge of Internet memes.

Another form of final project we often saw was using Pencil Code to re-create games, often pulling images and screenshots of the actual game as a way to make the game feel more authentic [32]. Students recreated classic games like Battleship, Pacman, and Pong, while also creating custom new games, like text-based fantasy adventures and zombie-hunting games that pulled in images from the Walking Dead TV show (Fig. 2C). Other students took advantage of Pencil Code's smart image finding logic to create choose your own adventure games that used images to tell the story, for example, telling a horror story using commands like: `wear anxious`, `wear abandoned house`, and `wear scary face` to tell the story.

In presenting this student work, we highlight specific ways that learners took advantage of the fact that there was no barrier between the Internet-based learning environment they were using and the Internet as a whole. By situating the learning environment online and making it easy to pull Internet resources into their project, learners were able to create personally meaningful projects, drawing direct connections between their own interests and the world of computing. This is not to say it is not possible to download and import images from web pages into standalone, offline environments, but rather that the amount of effort that takes is significantly more than simply typing `wear grumpy cat` into your program. The result was a set of projects filled with resources from the Internet and thus the larger world that the students live in.

5.2. Digital quilts and public projects

One of the powerful aspects of learning to program on the web, with the web, is that projects can easily be shared, browsed, and incorporated into other websites. In our introductory programming course, we designed the first assignment to take advantage of the web-based nature of both the programming environment and resulting programs. The assignment was for each student to design his or her own "quilt patch" – a program that draws a picture that somehow represented the program's author. This being the first assignment, the goal was for students to explore and try out different aspects of the tool. At the conclusion of the project, the researcher supporting the class created a web page to "stitch" the digital quilt patches together into a single web page (Fig. 3). Because the student projects were themselves websites, this was easily done using iFrames to create a web page of web pages. The next day in class, the teacher showed students the class quilt — which meant loading the quilt web page and watching the students' patches render in parallel. Students were then given a chance to guess who the author of each patch was, and students got to talk about the meaning of their patch.

Because the patches were written in Pencil Code, the resulting patches were just normal web pages, meaning their source code could be opened up and inspected. As part of showing the class their quilts, the teacher shared the URL to the quilt page with the students and showed them how to open up each patch individually, so students could see the code behind each drawing. In this way, students could not only see their classmates work but also see how other students authored their patches. This was particularly useful as some students had introduced sophisticated logic and employed advanced Pencil Code features to create their patches. The public and transparent nature of web pages provided an organic way for students to learn about programming, the capabilities of Pencil Code, and their classmates all at once. Through this assignment, various aspects of programming on the web, with the web come to the fore. Notably, the ability to blend the online projects together into a single page, the inherently public nature of the programs, and the ability to inspect projects to see how they were created. The quilt project was the first assignment of the year, based on feedback from the students and teacher, the "quilt" approach was used repeatedly throughout the course as a way for students to explore the programs written by their classmates.

On the last day of the Pencil Code units, in both classes, we provided opportunities during class time for students to share their projects. In the CT4G class, this meant students could volunteer to come up to the computer connected to the classroom projector and share their projects by typing in their URL (as mentioned previously with the Bieber project). In the programming classes, we used computer monitors that were set up around the room to let groups of students share their projects and play each other's games. Due to the ease of sharing projects, it was possible for students to describe how their project worked while their classmates played their games on their own computers. This sharing also extended beyond the classroom. As we learned in an interview with one of our students, when she showed her mom one of the interactive assignments she had completed, her mom responded: "*That's what you're learning? That's actually cool!*" The student continued, "*before* [my mom] *didn't really know what I was doing,*" then she went on to explain how her mom was familiar with the Internet, but not familiar with the field of computer science. In this instance, the projects being a website not only made it easy to share within the classroom but crossed the boundaries to the students' home, serving as a way to share computing coursework with a parent who was unfamiliar with programming. By being a website, the parent better understood what her daughter was learning as well as the cultural and professional value of the skills and concepts her daughter was developing.

6. Discussion

Bringing introductory programming tools online provides a number of cultural and conceptual supports that can be leveraged to help students engage with core computer science ideas. By linking the act of learning to program with the creation of websites that can easily be viewed and shared, introductory lessons take on a new form of authenticity both with respect to the tools being used and the products being created. The Web 2.0 model is grounded in the idea that the Internet is not a read-only resource, created by a small group of programming experts. Instead, the web is an emergent, collaborative creation of millions of users, each with their own voice. Aligning learning to program with participation in the World Wide Web highlights the cultural relevancy of computational literacy and links the discipline of computer science with the ubiquitous technologies the field has helped create.

From a pedagogical perspective, there are a number of strengths to programming on the Internet that align with the theoretical framework we bring to this work. By making the creation of websites the output of introductory activities, the act of programming is situated in a larger practice learners already engage in, browsing the Web, giving the approach what Papert called cultural syntonicity [2]. Further, websites are public artifacts that can be shared, viewed, and discussed, as demonstrated by the quilt activity — this public dimension can facilitate a sense of ownership and pride in the resulting outcome, as well as make it easy to provide feedback on projects written by peers, a powerful pedagogical practice [33, 34]. The graphical nature of websites also facilitates learning as programs written to dynamically create images, or graphics drawn on a website, give the author a visual representation that can facilitate interpreting the behavior of the programming constructs incorporated into the program [35].

A second important benefit of situating programming online is that it can help broaden participation in computer science. First, online learning environments are more accessible than off-line learning tools due to the breadth of devices (including tablets and smartphones) that can be used to access the learning environment. Further, programs started on one device can be continued on another and viewed and played with a third. The fact the programs are stored on the Internet makes it possible to progress in learning to program across various physical devices and from different locations. Projects started on a computer at a library, can be continued at a friend's house or in the school computer lab. In this way, hosting learning environments online can help pave a pathway into computing for learners who lack regular access to the same Internet-enabled device.

A second way online environments can help broaden participation in computing is by changing the nature of introductory computing activities to align with a broader range of interests [36]. The diversity of activities and interests that are mediated by the Web makes it a powerful and compelling context in which to situate introductory programming activities as can be seen in the projects we highlighted above, where Justin Bieber, Zombies from the Walking Dead, and school club websites lived alongside each other.

The benefits of online learning extend beyond the learner to also include teachers, system administrators, and the developers of the tools themselves. The fact the no special software needs to be installed to access the learning environment circumvents many challenges that arise when trying to install software on public or school-maintained computers. From the perspective of the creator and maintainer of the software, creating applications that run in web browsers has its own set of benefits, includes the ease of deploying bug fixes and being able to leverage the wealth of resources and libraries designed to support the development of web applications. From a research and evaluation standpoint, online tools are appealing as it is easy to gather data, which can then be used to inform future features and improvements to the learning environment.

While there are many reasons to situate introductory programming tools online, there are also drawbacks to the approach. These drawbacks have implications for teachers and learners as well as the designers and builders of such systems. From the perspective of the teacher and learner, foremost among the potential drawbacks of using online programming environments is the reliance on having Internet connectivity in order to use the tool. Hosting a learning environment online introduces an Internet dependency, which means they may not be suitable in environments where the Internet is not always available or is unreliable. Some online tools can be hosted locally which solves this specific issue but raises others, such as when and how projects connect to the larger web. A second related limitation is that in storing work remotely — the learner has little control over how and when it can be accessed. If the organization that maintains the learning environment takes it offline, the learner loses their work. Again, there are workarounds such as using robust cloud-based storage services such as DropBox or Microsoft OneDrive, but the integration with these platforms is rarely seamless and has some of the same potential pitfalls.

Aside from infrastructural limitations, there are also potential learning issues that might arise. For example, framing introductory programming tools as a way to make custom websites might give learners the impression that they can create their own Facebook or Twitter. If such unrealistic expectations emerge, students may become frustrated or disheartened when their sites do not have the polish and capabilities of the sites they are used to frequenting. Another drawback stems from the transparency inherent in a learning environment built around creating web pages. A tool where all the code is publically visible might not be ideal in formal learning settings as it becomes easy for learners to cheat by viewing the work of peers or others on the internet. A final related limitation the affects teachers and learners stems from the question of ownership of online material. While Pencil Code makes it extremely easy to bring in resources from the larger Internet, those resources may not always be free to use. While this can serve as an effective context to hold conversations around intellectual property and copyright issues, there are times this conversation is not desirable or there are no facilitators to lead that discussion.

There are also potential drawbacks to deciding to situate introductory programming learning tools on the Internet that affect the designers, builders, and maintainers of such systems. A primary concern is the reverse of the benefits of having access to the materials hosted on the Internet. Just as learners can incorporate pictures of their favorite athletes or audio clips of songs they like, so too can they incorporate inappropriate material into their projects. This might include pornographic images, racially insensitive and hate-filled material, or other content that is not appropriate for younger learners or educational contexts. Opening up a programming environment to the entirety of the web means that the potential for the incorporation of inappropriate material and thus, the maintainers of such platforms need to continually expend effort to make sure the content hosted and available to learners is appropriate for the audience. This requires continual vigilance. A second potential challenge for the creators of online programming tools is the lack of control over how the site is accessed. As new browsers, devices, and web standards emerge, the creators and maintainers of these online learning environments need to continually update the platform to ensure learners are always able to access their materials. While this is also true for stand-alone platforms that need to stay up to date with upgrades in operating systems, the rate of change of online technologies usually outpaces offline alternatives. While these challenges should give pause to designers when considering whether or not to design for the Internet, the huge upside to the online context suggests there are reasons to design for the cloud in spite of these drawbacks.

7. Conclusion

The Internet is increasingly becoming a context in which rich, meaningful learning takes place. In the case of programming, and computer science more broadly, situating introductory learning activities online, and providing accessible ways to draw in resources from the Internet, can result in authentic and engaging learning. Additionally, in making public web pages the output of introductory programming activities, it becomes easy to share, comment on, and customize the learning experience. By outlining various ways that the Web can serve as an engaging and effective context for introducing diverse learners to the field of computer science, this paper contributes to our understanding of the potential for the Internet to serve as a powerful, personally meaningful, culturally relevant, and fun way to introduce learners to the field of computer science. Through creating tools that help novices learn to program on the Internet, with the Internet, we hope to inspire the next generation of great software engineers and start-up entrepreneurs, while also serving as a means to prepare computationally literate citizens for the digital futures that await them.

Conflict of interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.ijcci.2019.01.001.

References

[1] A.A. diSessa, Changing Minds: Computers, Learning, and Literacy, MIT Press, Cambridge, MA, 2000.
[2] S. Papert, Mindstorms: Children, Computers, and Powerful Ideas, Basic books, New York, 1980.
[3] J.M. Wing, Computational thinking, Commun. ACM. 49 (2006) 33–35.
[4] D. Weintrop, E. Beheshti, M. Horn, K. Orton, K. Jona, L. Trouille, U. Wilensky, Defining computational thinking for mathematics and science classrooms, J. Sci. Educ. Technol. 25 (2016) 127–147, http://dx.doi.org/10.1007/s10956-015-9581-5.
[5] M. Guzdial, Programming environments for novices, Comput. Sci. Educ. Res. 2004 (2004) 127–154.
[6] C. Duncan, T. Bell, S. Tanimoto, Should your 8-year-old learn coding? in: Proc. 9th Workshop Prim. Second. Comput. Educ., ACM, New York, NY, USA, 2014, pp. 60–69, http://dx.doi.org/10.1145/2670757.2670774.
[7] C. Kelleher, R. Pausch, Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers, ACM Comput. Surv. 37 (2005) 83–137.
[8] D. Fields, M. Giang, Y. Kafai, Programming in the wild: trends in youth computational participation in the online scratch community, in: Proc. 9th Workshop Prim. Second. Comput. Educ., ACM Press, 2014, pp. 2–11, http://dx.doi.org/10.1145/2670757.2670768.
[9] R. Roque, Y. Kafai, D. Fields, From tools to communities: designs to support online creative collaboration in scratch, in: Proc. 11th Int. Conf. Interact. Des. Child., ACM, 2012, pp. 220–223, http://dl.acm.org/citation.cfm?id=2307130 (accessed 17-07-2015).
[10] Y.B. Kafai, Q. Burke, Connected Code: Why Children Need to Learn Programming, MIT Press, 2014.
[11] C. Greenhow, B. Robelia, J.E. Hughes, Learning, teaching, and scholarship in a digital age web 2.0 and classroom research: what path should we take now? Educ. Res. 38 (2009) 246–259.
[12] D.W. Shaffer, M. Resnick, "Thick" authenticity: new media and authentic learning, J. Interact. Learn. Res. 10 (1999) 195–215.
[13] B. DiSalvo, Graphical qualities of educational technology: using drag-and-drop and text-based programs for introductory computer science, IEEE Comput. Graph. Appl. (2014) 12–15.
[14] D. Weintrop, U. Wilensky, To block or not to block that is the question: students' perceptions of blocks-based programming, in: Proc. 14th Int. Conf. Interact. Des. Child., ACM, New York, NY, USA, 2015, pp. 199–208, http://dx.doi.org/10.1145/2771839.2771860.
[15] J. Lave, E. Wenger, Situated Learning: Legitimate Peripheral Participation, Cambridge Univ Pr, 1991.
[16] K. Kahn, TOONTALK REBORN re-implementing and re-conceptualising ToonTalk, in: Proc. Constr. 2014, Vienna, Austria, 2014, p. 8.
[17] U. Wilensky, NetLogo Web, Center for Connected Learning and Computer-Based Modeling, Northwestern University. http://www.netlogoweb.org, Evanston, IL, 2015.
[18] Scratch 2.0. (n.d.). In Scratch Wiki. Retrieved July 5, 2018, from https://en.scratch-wiki.info/wiki/Scratch_2.0.
[19] N. Fraser, Ten things we've learned from blockly, in: 2015 IEEE Blocks Workshop Blocks Beyond, 2015, pp. 49–50, http://dx.doi.org/10.1109/BLOCKS.2015.7369000.
[20] B. Harvey, J. Mönig, Bringing "no ceiling" to scratch: can one language serve kids and computer scientists? in: J. Clayson, I. Kalas (Eds.), Proc. Constr. Conf. 2010 Conf., Paris, France, 2010, pp. 1–10.
[21] A. Bruckman, Situated support for learning: Storm's weekend with Rachael, J. Learn. Sci. 9 (2000) 329–372.
[22] M. Resnick, B. Silverman, Y. Kafai, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, Scratch: programming for all, Commun. ACM. 52 (2009) 60.
[23] J.H. Maloney, M. Resnick, N. Rusk, B. Silverman, E. Eastmond, The scratch programming language and environment, ACM Trans. Comput. Educ. TOCE. 10 (2010) 16.

[24] D.A. Fields, M. Giang, Y.B. Kafai, Understanding collaborative practices in the Scratch online community: Patterns of participation among youth designers, in: N. Rummel, M. Kapur, S. Puntambekar (Eds.), CSCL 2013 Conf. Proc., Madison, WI, 2013, pp. 200–207.

[25] K.J. Harms, J.H. Kerr, M. Ichinco, M. Santolucito, A. Chuck, T. Koscik, M. Chou, C.L. Kelleher, Designing a community to support long-term interest in programming for middle school children, in: Proc. 11th Int. Conf. Interact. Des. Child., ACM, New York, NY, USA, 2012, pp. 304–307, http://dx.doi.org/10.1145/2307096.2307152.

[26] R. Lerner, S.T. Levy, U. Wilensky, Encouraging collaborative constructionism: principles behind the modeling commons, in: I. Kalas J. Clayson (Ed.), Proc. Constr. Conf., Paris, France, 2010, pp. 10–14.

[27] D. Bau, D.A. Bau, M. Dawson, C.S. Pickens, Pencil code: block code for a text world, in: Proc. 14th Int. Conf. Interact. Des. Child., ACM, New York, NY, USA, 2015, pp. 445–448, http://dx.doi.org/10.1145/2771839.2771875.

[28] D. Weintrop, N. Holbert, From blocks to text and back: programming patterns in a dual-modality environment, in: Proc. ACM SIGCSE Tech. Symp. Comput. Sci. Educ., ACM, New York, NY, USA, 2017, pp. 633–638, http://dx.doi.org/10.1145/3017680.3017707.

[29] D. Weintrop, Modality Matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms, (Ph.D Dissertation), Northwestern University, 2016.

[30] K. Orton, D. Weintrop, E. Beheshti, M. Horn, K. Jona, U. Wilensky, Bringing computational thinking into high school mathematics and science classrooms, in: Proc. ICLS 2016, Singapore, 2016, pp. 705–712.

[31] C. Brady, K. Orton, D. Weintrop, G. Anton, S. Rodriguez, U. Wilensky, All roads lead to computing: making, participatory simulations, and social computing as pathways to computer science, IEEE Trans. Educ. 60 (2016) 1–8, http://dx.doi.org/10.1109/TE.2016.2622680.

[32] D. Weintrop, U. Wilensky, Keeping it old school: classic video games as inspiration for modern student programs, in: Proc. 11th Games Learn. Soc. Conf., Madison, WI, 2015.

[33] S. Papert, I. Harel, Situating constructionism, in: S. Papert I. Harel (Ed.), Constructionism, Ablex Publishing Corp., Norwood N.J., 1991, pp. 1–11.

[34] Y.B. Kafai, Minds in Play: Computer Game Design as a Context for Children's Learning, Routledge, 1994.

[35] T.L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, Exploring the role of visualization and engagement in computer science education, in: ACM SIGCSE Bull., 2002, pp. 131–152.

[36] J.J. Ryoo, J. Margolis, C.H. Lee, C.D. Sandoval, J. Goode, Democratizing computer science knowledge: transforming the face of computer science through public high school education, Learn. Media Technol. 38 (2013) 161–181, http://dx.doi.org/10.1080/17439884.2013.756514.