Effects of Code Representation on Student Perceptions and Attitudes Toward Programming

Jeremiah Blanchard
Department of CISE
University of Florida
Gainesville, FL, USA
jjb@eng.ufl.edu

Chistina Gardner-McCune
Department of CISE
University of Florida
Gainesville, FL, USA
gmccune@ufl.edu

Lisa Anthony
Department of CISE
University of Florida
Gainesville, FL, USA
lanthony@cise.ufl.edu

***Abstract*— Text languages are perceived by many computer science students as difficult, intimidating, and/or tedious in nature. Conversely, blocks-based environments are perceived as approachable, but many students see them as inauthentic. Bidirectional hybrid environments provide textual and blocks-based representations of the same code, thereby offering students the opportunity to seamlessly transition between representations to build a conceptual bridge between blocks and text. However, it is not known how use of hybrid environments impacts perceptions of programming. To investigate, we conducted a study in a public middle school with six classes (n=129). We found that students who used hybrid environments perceived text more positively than those who moved directly from blocks to text. The results of this research suggest that hybrid programming environments can help to transition students from blocks to text-based programming while minimizing negative perceptions of programming.**

Keywords—Computer Science Education, blocks-based programming environments, programming languages, novice programmers, hybrid programming environments.

I. Introduction

Programming instruction has traditionally made use of text-based production languages [1] (i.e., Python, C/C++, and Java). This has the benefit of anchoring instruction in languages used in industry [2], but it presents difficulties for new students of computer science. Working in text has been found to increase *cognitive load* of students compared to other types of programming environments [3]. In focus group interviews, novice students have noted that they perceive text-based programming languages as difficult, intimidating, and/or tedious in nature [4]. The initial experiences students have while programming affect their perceptions of their own ability to program [5] and of programming in general, which plays a role in cultivating or discouraging students' interest in computing fields [2].

Many educators use blocks-based programming environments to introduce students to programming without the challenges of text-based programming, particularly in grades K-12 [6], [7]. The ease of programming in blocks has helped to increase the introduction of programming to a large number of students in K-12 [8]–[10]. While many students perceive block-based environments as easier than text to use, they also see blocks as inauthentic since these languages are not used in the computing industry at large [2], [11]. In addition, it has been reported that some students have difficulty transitioning from blocks-based environments to text-based languages [11]. As a result, CS Educators and researchers have expressed the need for tools that help students transition from blocks-based to text-based languages used in industry and also to seek ways to counter the perceptions of the inauthenticity of blocks-based programming languages [3]. Some members of the CS education community believe that hybrid blocks-to-text systems, which represent programs in *both* text and blocks-based representations, provide an opportunity for students to see the connection between these representations [12]. Further, hybrid environments may alleviate perceptions of inauthenticity of blocks-based representations [3]. Identifying how hybrid environments affects students' perceptions of programming representations can help instructors develop more effective introductory programming curricula and guide their selection of programming environments.

To investigate how bidirectional hybrid environments affect students transitioning from blocks to text, we conducted a three-condition study in a public school with six classes (n=129). Two classes (n=35) moved from blocks directly into text, while two other classes (n=48) used a bidirectional hybrid programming environment [13] before moving to text-only programming. The last two classes (n=46) acted as the control and used a text-based programming environment for the duration of the study. We measured participant learning via a custom assessment that we designed based on the SCS1 [14] and Computer Science Principles sample exam questions [15]. We also surveyed students on their perceptions of programming and blocks-based programming as compared to text-based programming (and vice-versa). Our results show that participants perceived text positively more often when they transitioned from blocks to text via a bidirectional hybrid environment as compared to those who worked only in text or moved directly from blocks to text. These results suggest transitioning from blocks to text via a hybrid environment can serve as a bridge from blocks-based to production languages and helps students establish a level of comfort with text-based languages. Our work will assist educators in transitioning students from blocks to text while minimizing issues of negative programming perceptions.

II. Background

Early hybrid programming environments provided a unidirectional translation from blocks to text [16]. This allowed students to program in blocks and then see the corresponding text representation of their code. Thus, early hybrid environments could provide the affordances of blocks-based programming, such as syntax scaffolding and usability, while linking blocks to text, supporting authenticity [3]. For example, Alice 3 provides syntactically correct, executable Java text from its visual constructs [16], [17].

978-1-7281-0810-0/19/$31.00 ©2019 IEEE

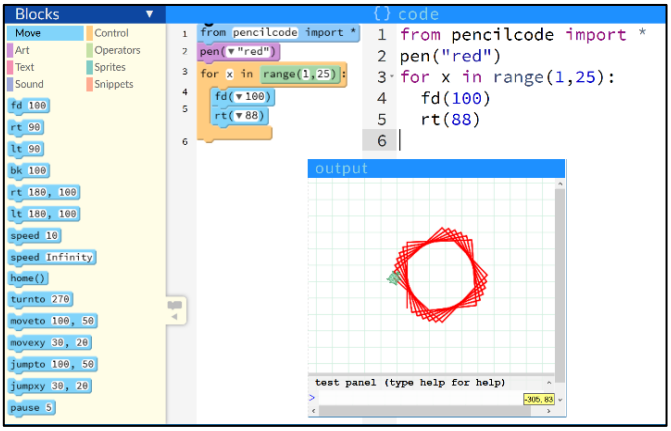

Fig. 1. Python variant of Pencil Code showing blocks and text modes.

While early hybrid environments allowed translation in only one direction, bidirectional hybrid environments allow translation in both directions (blocks-to-text and text-to-blocks). For example, Pencil Code and its Droplet Editor (Fig. 1) allow the seamless transition between blocks and text and vice versa at the click of a button [18]. Thus, bidirectional platforms may be effective in enabling students to more smoothly transition from a visual environment to a text language at their own pace instead of switching all at once [13]. Similar to unidirectional hybrid environments, bidirectional hybrid environments may also be able to address perceptions of blocks-based representations as inauthentic [11] by clearly linking blocks and text based representations of the same code.

Finally, hybrid environments may be able to overcome actual and/or perceived difficulty in learning and use of text languages by providing a practical and usable environment that allows quick transition between blocks and text representations of the same program semantics. Development of bidirectional environments is relatively recent, thus there is limited research into their effectiveness, their impacts on perception, and the role they may be able to play in education.

III. Methods

In this paper we aim to answer the question **How do bidirectional hybrid environments impact student perceptions of programming?** In designing a study to answer this question, we were particularly interested in examining student confidence in their own ability to program, student perceptions of text- and blocks-based environments, and the impacts of their perceptions on their learning.

We ran our study at a large public middle school in the southeastern United States in 2017. The study involved participants in a technology course (with six class periods) under the supervision of a single instructor. Prior to participation in the study, the course instructor had planned to offer programming instruction as part of the curriculum of the course. We partnered with the teacher to use curriculum we designed and our study framework to offer this instruction. The curriculum focused on variables, loops, selection, and functions. Of 24 school days included in the study, nine were dedicated to state standardized assessments, leaving 15 days of instruction and three days of surveys and assessments for this study. Depending on the testing schedule, participants received multiple days of CS instruction per week. Each class period was 38-46 minutes, for a total of 12 contact hours. The classroom teacher and one researcher, the first author, co-instructed the course during the instructional period.

A. Participants

We conducted our study with six classes of eighth-grade students. Before the study began, participants took home an IRB-approved letter describing the study's purpose and informing guardians of their rights to opt their child out of the study. We also asked students on the first day if they voluntarily assented to participate in the study. No compensation was provided. Of 158 students in six classes, 129 students agreed to participate in the study. Students who did not agree to participate received the same instruction and in-class programming assignments as study participants, but did not take study surveys.

We obtained demographic data by self-report. The participants ranged in age from 12 to 16 years old at the time the study was conducted: 86.0% (n=111) were 13 to 14 years old; 2.3% (n=3) were 12; and 5.4% (n=7) were 15 to 16 years old. Eight participants did not provide their age. 39.5% (n=51) of participants identified as female, while 51.9% (n=67) identified as male; one participant (0.8%) identified as gender neutral. Ten participants did not provide a gender. The classes were ethnically diverse. Of participants reporting only one ethnic background, 25.6% (n=33) identified as white; 30.2% (n=39) as Hispanic/Latino; 4.7% (n=6) as black or African American; 4.7% (n=6) as Asian; and 1.6% (n=2) as Native Hawaiian or Pacific Islander. 29.5% (n=38) of participants reported multiple ethno-racial backgrounds. Five did not note a background.

B. Study Design

We developed a Python variant [19] for Pencil Code [18] for this study. Classes were taught using three tailored versions of the Pencil Code environment. We subdivided the six classes into three condition groups of two classes each: Blocks, Hybrid, and Text. Figure 2 summarizes the amount of time each condition spent in blocks, hybrid, or text mode and on assessments. Participants in the Blocks condition spent eight days using a blocks-based environment, followed by seven days using text; those in the Hybrid condition spent four days in blocks, five days in the bidirectional hybrid environment, and six days in text; and participants in the Text condition used a text-only variant of Pencil Code for 15 days. Note that text syntax was available to all students at all times due to the design of the Pencil Code blocks, which presents the full text syntax of the constructs on the blocks. Three days were dedicated to assessments and surveys throughout the study.

C. Data Collection

1) Surveys. At the end of the study, we asked participants about their perceptions of blocks and text (see Table I) as well as several other questions that are not analyzed in this paper. Survey questions used a 7-point Likert scale to rate agreement/disagreement ("Strongly Disagree", "Disagree", "Somewhat Disagree", "Neutral", "Somewhat Agree", "Agree", "Strongly Agree"). Each such question was paired with a free response prompt: "Why do you feel this way?" All of the students who agreed to participate in the study (n=129)

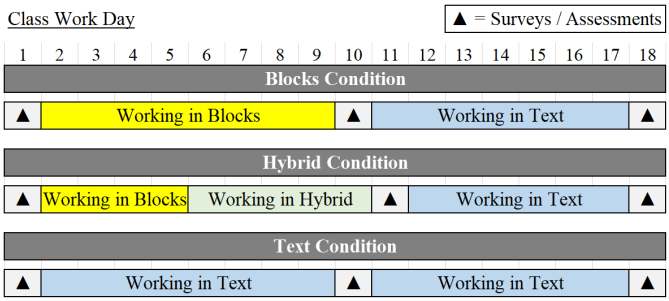

Fig. 2. Timeline of time spent in text / hybrid / blocks modes by condition.

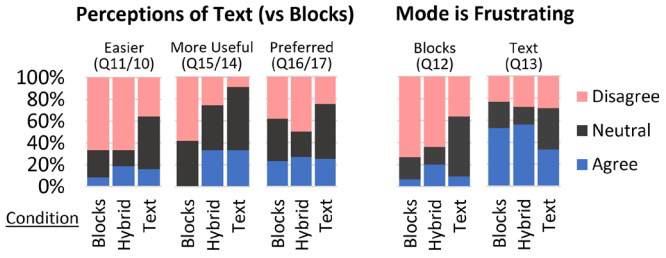

Fig. 3. Distribution of survey Likert responses.

took the initial (pre-study) survey. Due to class absences, of the 129 participants, 38.8% (n=50) participated in the mid-survey, and 57.4% (n=74) participated in the post-survey.

2) Assessments. We assessed participants' learning using a custom assessment we developed based on questions from the SCS1 instrument [14] and sample questions in the Computer Science Principles AP course and exam description [15]. Each question focused on a different programming concept, with an equal number of questions assessing for-loops, while-loops, selection (if-else), and functions. Blocks-based and text-based isomorphic variants of each question were developed at multiple levels of difficulty. As most students (n=87) had prior experience in blocks, the initial assessment used only blocks, while the final assessment used only text. The mid-assessment was dependent upon condition, with blocks condition participants receiving a blocks-only assessment, text condition participants receiving a text-only assessment, and hybrid condition participants receiving a mixed assessment.

This paper focuses on the survey data and what it reveals about student attitudes and perceptions of programming between these three different conditions. Our future work will involve analysis of the learning assessments and their correlation with their perceptions.

D. Data Anaylsis

To examine the impact of the programming environment on students' perception of blocks and text, we analyzed the Likert responses and free response question answers. We converted all Likert responses to numeric values (1 to 7), inverted the value for blocks-preference responses, and calculated the midpoint of the two variants for question pairs. We grouped responses with midpoints of 1-3.9 as "disagree", 4.0 as "neutral", and 4.1-7 as "agree". We present the proportion of students who agreed, disagreed, or were neutral for each question in Figure 3. The *n* varies per question since not all students opted to answer all questions. We qualitatively coded the free responses to identify themes related to participants' perceptions of blocks and text programming. Using an inductive qualitative coding approach [20], the first author created and assigned codes to each response. Each response fell into one of these themes: Pro-Text, Pro-Blocks, Anti-Text, Anti-Blocks, Neutral. To compute interrater reliability for each question and response code [21], a second researcher coded 16% of the responses. The average agreement between coders was Cohen's kappa = 0.7845, which is characterized as substantial agreement [22].

TABLE I. QUESTIONS COMPARING BLOCKS & TEXT PROGRAMMING

Q11	I think programming in text is easier than programming in blocks.
Q10	I think programming in blocks is easier than programming in text.
Q12	I think programming in blocks is frustrating or hard.
Q13	I think programming in text is frustrating or hard.
Q15	I think learning to program in text is more useful than blocks.
Q14	I think learning to program in blocks is more useful than text.
Q16	I would prefer to program using text as opposed to blocks.
Q17	I would prefer to program using blocks as opposed to text.

IV. FINDINGS

We examined the distribution of participant responses to Likert scale questions regarding participants' perceptions of text as easy (Q10/Q11) and frustrating (Q13), and also the coded responses to the accompanying free response questions.

Hybrid condition participants most often rated text as easier than blocks compared to blocks condition participants (Q10/Q11, Likert). On the final survey, 18.5% (n=5) of those in the hybrid condition identified text as easier than blocks, while 14.8% (n=4) were neutral and 66.7% (n=18) disagreed. In contrast, fewer participants in the blocks condition agreed that text was easier than blocks (agree: 8.3% (n=1); neutral: 25.0% (n=3); disagree: 66.7% (n=8)). Text condition students rated text representations as easier about as frequently as hybrid students, but disagreed less often (agree: 16.0% (n=4); neutral: 48.0% (n=12); disagree: 36.0% (n=9)).

Hybrid condition participants perceived text more favorably than blocks condition participants (all free responses). For every one of the text-blocks comparison questions we asked, hybrid condition participants were 1) more often pro-text and 2) less often anti-text than their blocks condition counterparts. Hybrid condition participants also frequently responded by comparing the environments when giving a neutral response.

One common reason given by hybrid condition participants for why they liked text was that they felt it helped them make rapid progress, with students noting: *"…it is a lot faster and easier to understand."* [H002] and *"…I think text is faster and makes it easier to change the code"* [H024]. These responses suggest that hybrid condition participants developed an appreciation for the benefits of text in terms of efficiency in programming. Another common reason cited by hybrid condition participants for preferring text over blocks was that they found text to be more organized and easier to debug: *"It's faster for me to recognize the error in my code when looking at text and it is easier to organize"* [H002]. They also felt text offered more flexibility than blocks: *"text is more free in what you can do while blocks have very restrictive ways of coding"* [H099]. Hybrid condition participants who perceived text less favorably than blocks cited syntax issues as their biggest challenges: *"because when you [are] programming in text there*

is a million ways you can mess up the coding. and it[']s not always easy remembering the codes" [H042].

Many responses given by hybrid condition participants were comparative, noting pros and cons of a particular mode. One hybrid student said, *"because… [blocks are] easier but at the same time you need to get used to it [text]"* [H126], while another noted that *"I think that they both have their advantages"* [H065]. The comparisons expressed in these responses are evidence of a more nuanced view of programming representations, weighing the benefits and drawbacks of blocks and text.

On the other hand, participants in the blocks condition were more negative about text programming, frequently mentioning syntax and detail issues that they felt got in their way: *"it takes to[o] long to write and any little mistake can mess up the whole thing"* [B047]. Another blocks condition participant said, *"Any small mistake will make it say 'script error'"* [B043]. From these responses, we see that blocks participants primarily focused on the difficulties that text presented and were not able to recognize the strengths of text in terms of organization and flexibility that hybrid participants noted.

Hybrid and blocks conditions participants both found text frustrating, unlike text condition participants (Q13, Likert and free-response). 56.0% (n=14) of hybrid condition participants agreed that programming in text was frustrating or hard, while 16.0% (n=4) were neutral and 28.0% (n=7) disagreed. Similarly, the majority of blocks condition participants agreed that text was frustrating or hard (agree: 52.9% (n=9); neutral: 23.5% (n=4); disagree: 23.5% (n=3)). Meanwhile, only 33.3% (n=8) of text condition participants agreed that text is frustrating or hard (neutral: 37.5% (n=9); disagree: 29.2% (n=7)).

Many text condition participants described feeling comfortable using text despite the challenges they noted: *"I feel it's really easy, I just need a little more practice"* [T082]. In contrast, hybrid and blocks condition participants mentioned similar obstacles and were more discouraged, rating text as more frustrating. One blocks condition participant noted that *"[text is more frustrating than blocks because] the text has to be perfect"* [B040] and one hybrid-condition participant said, *"You have to beware of many errors because when you do it wrong you have to figure out where you messed up and it takes a while"* [H122]. These responses show that participants in all conditions referred to experiencing obstacles in using text related to syntax. Text condition participants framed them as challenges to master, while hybrid and blocks condition participants interpreted them as impediments that limited their progress. It is notable that text students spent the entire study within the text environment, and thus had more time to achieve a high level of comfort in text.

V. Discussion

In this study, participants who used hybrid environments rated text easier to use when compared to those who moved directly from blocks to text. Both hybrid and blocks condition participants experienced more frustration in text. However, we also found that, in general, hybrid condition participants held positive perceptions of text more frequently across questions regarding difficulty, frustration, usefulness, and preferred mode of programming, compared to blocks condition participants.

Blocks participants in our study had a less favorable view of text than hybrid and text participants. Responses may reflect the frustration of moving directly from blocks to text, suddenly losing the scaffolding on which they had come to depend, which was also supported by our classroom observations during the study. After moving to text, blocks students especially expressed frustration related to usability and increased errors.

Hybrid participants expressed more positive views of text representations, overall, than their counterparts in the blocks condition. Many hybrid participants expressed positive views of working in text, stating that text was easier to understand and helped support their learning, while others described it as fun – suggesting that they had developed a level of comfort in text programming. Classroom observations during the study sessions confirmed that students frequently flipped back and forth between blocks and text – taking advantage of the scaffolding that bidirectional hybrid environments provide. This allowed each participant to transition at their own pace, making the transition from blocks to text less jarring and more inviting. This complements prior work showing that students in hybrid environments often switch between blocks and text when new constructs are introduced [18]. The self-paced transition is particularly important as increase in confidence is one of the major motivations for creating visual (and especially blocks-based) languages [23], and our findings suggest hybrid environments may help achieve educational goals of blocks-based environments.

Perceptions of programming can impact perseverance in the field by newcomers [24]; our study demonstrates how those perceptions differ based on the tools used to transition between blocks and text. These findings suggest that educators can reduce the hurdles and frustrations students face when moving from blocks to text by utilizing an environment that bridges representations. By developing approaches to computer science instruction that reduce perception of difficulty and frustration, and improve perceptions of usefulness, we remove common obstacles that students face when first engaging with programming and transitioning to text-based programming.

VI. Limitations & Future Work

Our study was designed to be as close as possible to a typical classroom experience for the participants, and such studies come with logistical challenges and limitations that impacted the scope of our study, including (a) study duration (12 contact hours per participant over 15 days plus three days of assessments), (b) population (only 8th grade students), (c) logistical challenges (absences, study drop-outs, and non-compliance), and (d) measurements (gathering only post-survey data for some questions without pre-survey data). In particular, absences impacted a non-trivial portion of participants due to end of year activities and assemblies which affected survey participation. Also, perceptions of the hybrid experience could be influenced by affordances of the specific language (Python) and/or environment (Pencil Code) we used and should be examined in future work. In this paper we only report the results from some of our survey questions and did not present the assessment data; we plan to examine these data in future work.

ACKNOWLEDGEMENTS

We extend our thanks and appreciation to Kent Wenger for opening his classroom and allowing us to work with students in his technology classes. We also thank Pedro G. Feijóo-García for his time and effort in coding responses for inter-rater reliability.

REFERENCES

[1] A. L. Tharp, "Selecting the 'right' programming language," in *Proc. 13th SIGCSE technical symposium on Computer science education*, 1982, pp. 151–155.

[2] B. Disalvo, "Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science," *IEEE comput. graph. appl.*, vol. 34, no. 6, pp. 12–15, 2014.

[3] D. Bau, J. Gray, C. Kelleher, J. Sheldon, and F. Turbak, "Learnable programming: blocks and beyond," *Commun. ACM*, vol. 60, no. 6, pp. 72–80, 2017.

[4] A. Begel and E. Klopfer, "StarLogo TNG: An introduction to game development," *J. E-Learning*, vol. 53, p. 146, 2007.

[5] M. Biggers, A. Brauer, and T. Yilmaz, "Student perceptions of computer science: a retention study comparing graduating seniors with CS leavers," in *Proc. 39th SIGCSE technical symposium on Computer science education*, 2008, pp. 402-406.

[6] D. Weintrop and U. Wilensky, "Bringing blocks-based programming into high school computer science classrooms," paper presented at *Annual meeting of the American Educational Research Association*, 2016.

[7] J. Goode and J. Margolis, "Exploring computer science: a case study of school reform," *ACM Trans. Comput. Educ.*, vol. 11, no. 2, 2011.

[8] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts," *J. Comput. Sci. Coll.*, vol. 15, no. 5, pp. 107–116, 2000.

[9] J. Maloney, M. Resnick, and N. Rusk, "The Scratch programming language and environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, pp. 1–15, 2010.

[10] S. Goschnick, "App review: ScratchJr (Scratch Junior)," *Int. J. People-Oriented Program.*, vol. 4, no. 1, pp. 50–55, 2015.

[11] D. Weintrop and U. Wilensky, "To block or not to block, that is the question: students' perceptions of blocks-based programming," in *Proc. 14th international conference on interaction design and children*, 2015, pp. 199–208.

[12] N. C. C. Brown, J. Mönig, A. Bau, and D. Weintrop, "Panel: Future Directions of Block-based Programming," in *Proc. 47th SIGCSE technical symposium on Computing science education*, 2016, pp. 315–316.

[13] D. Bau, "Droplet, a blocks-based editor for text code," *J. Comput. Sci. Coll.*, vol. 30, no. 6, pp. 138–144, 2015.

[14] M. C. Parker and M. Guzdial, "Replication, validation, and use of a language independent CS1 knowledge assessment," in *Proc. 2016 ACM conference on international computing education research*, 2016, pp. 93–101.

[15] "AP computer science principles," 2016. [Online]. Available: https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap-computer-science-principles-course-and-exam-description.pdf. [Accessed: 01-Oct-2017].

[16] P. M. Mullins and M. Conlon, "Engaging students in programming fundamentals using Alice 2.0," in *Proc. 9th ACM SIGITE conference on Information technology education*, 2008, pp. 81–88.

[17] W. Dann, D. Cosgrove, D. Slater, and D. Culyba, "Mediated transfer: Alice 3 to Java," in *Proc. 43rd SIGCSE technical symposium on Computer science education*, 2012, pp. 141–146.

[18] D. Bau, D. A. Bau, C. S. Pickens, and M. Dawson, "Pencil Code: block code for a text world," in *Proc. 14th international conference on interaction design and children*, 2015, pp. 445–448.

[19] J. Blanchard, "Hybrid environments: A bridge from blocks to text," in *Proc. 2017 ACM conference on international computing education research*, 2017, pp. 295–296.

[20] C. Auerbach and L. B. Silverstein, "Qualitative data: An introduction to coding and analysis," in *Qualitative data: An introduction to coding and analysis*, NYU press, 2003, pp. 31–87.

[21] K. L. Gwet, "Computing inter-rater reliability and its variance in the presence of high agreement," *Br. J. Math. Stat. Psychol.*, vol. 61, no. 1, pp. 29–48, 2008.

[22] A. J. Viera and J. M. Garrett, "Understanding interobserver agreement: The kappa statistic," *Fam. Med.*, no. May, pp. 360–363, 2005.

[23] A. McGettrick, L. Cassel, M. Guzdial, and E. Roberts, "The current crisis in computing: What are the real issues?," in *Proc. 38th SIGCSE technical symposium on Computer science education*, 2007, pp. 329–330.

[24] B. DiSalvo, M. Guzdial, A. Bruckman, and T. McKlin, "Saving face while geeking out: Video game testing as a justification for learning computer science," *J. Learn. Sci.*, vol. 23, no. 3, pp. 272–315, 2014.