# Measuring the Computational in Computational Participation: Debugging Interactive Stories in Middle School Computer Science

Chris Proctor, Stanford Graduate School of Education, cproctor@stanford.edu

**Abstract:** An equitable implementation of K-12 computer science must support inclusive literacy practices, but it must also develop concrete skills. This study analyzes the extent to which a computer science curriculum based on digital storytelling helped students become more effective at debugging. Prior research has developed digital storytelling as a medium for computational participation, but few studies have reported detailed results on growth in computer science skills. Meanwhile, research on debugging has tended not to address sociocultural factors. This study, conducted over four months of a middle-school computer science course using interactive storytelling, analyzed student reading, writing, and debugging practices based on approximately 1000 story edits and user behavior collected from the platform's logs. The results suggest that literacy-based computer science education using digital storytelling can be productive for developing skills such as debugging.

## Introduction

Driven by the economic opportunities and pervasive societal impacts of computing, computer science (CS) is rapidly becoming a mainstream subject in primary and secondary education. Computational thinking is developing as a set of ideas and practices to be taught across the curriculum. In either case, the interdisciplinary connections to STEM subjects are clear and compelling (Weintrop, et al., 2014), while much less is said about how CS might support the core concerns of the humanities and social sciences. Indeed, when CS is defined narrowly as a collection of facts and skills about solving problems with computers, the subjects may not have much common concern. However, youth today engage in diverse and complex literacy practices with digital media (Ito, et al., 2010), relying to various extents on the computational aspects of these media to engage in computational participation (Burke, O'Byrne, & Kafai, 2016). If K-12 CS were contextualized within these literacies, its central concepts and practices could become an essential part of diverse youth cultures reading, writing, and analyzing digital texts.

Recognizing this opportunity (particularly for youth who do not see their identities and cultures represented in the world of CS), there have been numerous efforts to incorporate digital storytelling into school. Like projects which embed computation into animation (Resnick, et al., 2009) and e-textiles (Buechley, 2006), digital storytelling allows learners to learn programming and encounter powerful ideas from computer science using media which already mediate their literacy practices. Ware & Warschauer (2005) explored the potential for digital storytelling to transcend the divide between school and informal spaces for youth marginalized by race and social class. Kelleher & Pausch (2007) conducted digital storytelling workshops with middle-school girls using a modified version of Alice, finding that the opportunity for self-expression, sharing, and identity development provided motivation for learning to program. Burke & Kafai (2010) drew on parallels between programming and writing to explore how digital storytelling might support growth in each. Proctor and Blikstein (2019) analyzed how the computational elements of interactive stories could function rhetorically and support the development of critical perspectives.

However, some computer scientists argue that sociocultural definitions of K-12 computer science, such as those above, are unworkably vague and impossible to assess (Denning, 2017). This critique demands a response, particularly as it aligns with broader arguments that specific skills are best taught in a context that minimizes cognitive load (Kirschner, Sweller, & Clark, 2006). Must the literacy-based approaches described above involve a tradeoff in terms of how well students learn foundational skills and concepts? Building on prior work developing digital storytelling as a fruitful medium for computational literacies, this study analyzes whether such an environment can also effectively support specific computational practices such as debugging. This study's research questions are:

1. Does writing interactive stories support development of debugging practices?
2. Does reading other stories support development of debugging practices?

## Background

 © ISLS

## Text-based interactive storytelling

Interactive storytelling can be distinguished from the broader category of digital storytelling by the use of programming to implement nonlinear single-player games in which the player becomes a character. Text-based interactive stories are particularly effective settings for using computational elements for rhetorical effect in the service of representing and critically analyzing social realities (Proctor & Blikstein, 2019; Proctor & Garcia, 2019). In contrast to primarily-visual storytelling platforms such as Scratch and Alice, writing is singularly important for narrative, representation, and analysis of subjective phenomena in popular culture and in the humanities and social sciences. One example of the transmedia possibilities of text-based interactive storytelling is a high school sociology class in which students wrote interactive stories exploring the use of power in social interactions. The stories' use of programming allowed them to create replayable models of social situations in which readers could explore the consequences of different interactional choices.

The implementation used in this study is a web application called Unfold Studio in which stories' source code is presented side-by-side with a running version. Unfold Studio has social affordances such as the ability to publish stories, browse and read other authors' stories, and a feed showing events related to an author's stories and other authors she follows. Stories are written in a programming language called Ink (Inkle, 2016) and compiled every time an author saves her work. The story excerpt shown in Figure 1 illustrates the syntactical elements analyzed in this study. Chunks of the story are defined as knots (**`=== hotel ===`**), which are linked together via diverts (**`->`**). Knots typically end by presenting the player with choices (**`+`**) of what should happen next. Constructs within curly braces allow variables to influence the content and choices shown to the player. When stories contain errors, explanatory error messages are shown in the space that would have been taken up by the running story.

```
Welcome to California
+ I am a 14 year old girl.
      -> student
+ I am a 50 year old man.
    -> teacher

=== teacher ===
As an adult you have special
priviledges.
+ Fly to San Fransisco
    -> enter

=== student ===
You can show your parent you
can do something on your own!
+ Fly to San Fransisco for a
short independace trip.
    -> enter

=== enter ===
You made it to San
Fransisco!!!
+ Go to the hotel
    -> hotel

=== hotel ===
{teacher > 0:
      The Man at the front
desk says, "Hello Sir, do you
have a reservation for a
room?"
      -> room
 - else:
      The Man at the front
desk says, "Why are you here?
Your dont have a reservation!
Where are your parents? Go
away child!" Oh no! Your
stuck to sleep on the steets!
      -> streets
}
```

Figure 1. Excerpt from an interactive story written in the Ink programming language.

## Debugging

This study analyzes the association between writing interactive stories and performance on a debugging task. McCauley, et al.'s (2008) review of educational literature on debugging defines debugging as part of a response to some kind of breakdown in a programmer's plan for reaching a goal. After testing reveals that a breakdown has occurred, debugging involves ‘‘find[ing] out exactly where the error is and how to fix it.’’(p. 68) The K-12 Computer Science Framework (2016) uses a similar definition and adopts testing and debugging as one of seven core practices in computer science (p. 81). Although debugging has historically not been emphasized in computing education, it occupies a substantial portion of professional programmers' time (Beller, et al., 2017). Bugs may occur at the level of syntax (the program cannot be compiled), semantics (the program crashes at runtime), or logic (the program works, but not as expected). The available data limits this study to an analysis of bugs in syntax.

Debugging is a particularly interesting skill to study in an interactive storytelling context. On the one hand, prior research has found debugging to be a distributed social practice, which suggests a literacy-oriented approach could be effective in developing students' debugging skills. Flood, et al. (2018) found that learning to debug resembles enskilment in which "a newcomer is supported in appreciating and using the affordances of their environment" (p.1). Piorkowski, et al. (2013), found that during debugging, professional programmers spend half their time in information-foraging behaviors such as reading other programs, reading documentation, and searching online. Multiple studies have found comprehension of the program being debugged (and understanding of its goal) to be an important factor in debugging

success (McCauley, et al., 2008). This invites comparisons to the importance of reading comprehension for fixing grammatical errors in writing (Weaver, 1996).

On the other hand, foundational research on debugging identified misconceptions which could be exacerbated by a literacy-based approach. Bonar and Soloway (1985) found that novices misunderstand programming as writing. In particular, novices tended to inappropriately apply natural language meanings to words such as "while." Pea (1986) saw this as an instance of a more general superbug: novices' misconception that computers reason about programs and interpret them intelligently, instead of following them mechanistically. It is plausible that text-based interactive storytelling, in which authors blend prose with code to implement discourse scenarios, could unproductively mingle the ways humans interpret text and the ways machines interpret code. However, it is also plausible that these misconceptions could actually be ameliorated by casting programming as reading and writing. When there is real authorial meaning motivating the program, and a real audience to whom it is addressed, it could be the case that students better understand the computer's mediating role, rather than misunderstanding it as co-author or interlocutor.

## Methods

### Context

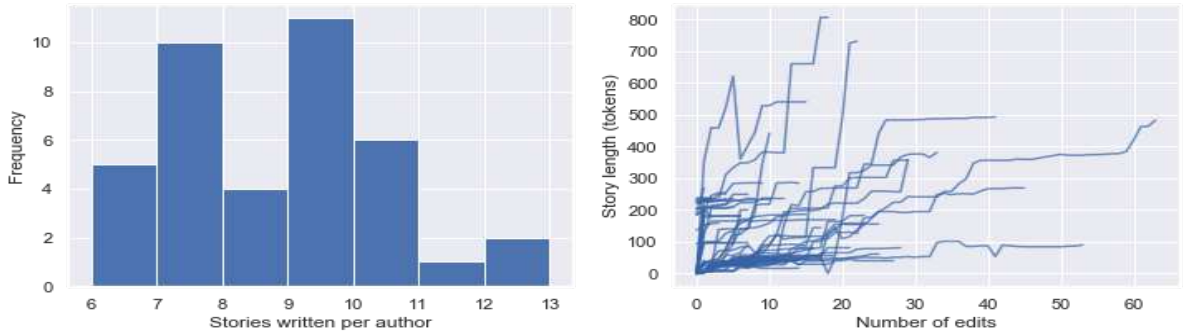
The study took place over four months in a private all-girls middle school in the western United States. The participants, in seventh grade, were enrolled in a computer science class which met twice weekly for a 90-minute block period. 40 students out of a cohort of 67 consented to participate in the study. Students were asked to self-identify with respect to race and gender. 92.5% affirmatively identified as female; 37.5% identified as white, 27.5% as of mixed race, 20% as Asian, 5% as hispanic, and 10% declined to state. While no data on socioeconomic status were available, the school is located in an affluent area and charges substantial tuition while also offering full or partial scholarships to many of its students.

During the period of the study, the students' computer science class followed the curriculum developed by Proctor & Blikstein (2019). Students were introduced to increasingly complex computational concepts and programming syntax and asked to use them in open-ended story prompts. During class time, students were free to work on their stories, read their peers' stories, and to seek help from peers or the teacher. The study's author worked with the teacher to plan the unit and provided technical support, but was not involved in teaching.

### Data sources

#### Stories

There are 327 stories from 39 authors with an average of 92 lines ($\sigma = 84$) and 431 tokens ($\sigma = 400$), where tokens include individual words as well as syntactical elements. A snapshot was captured every time a student saved her story. Saving a story was a necessary step in recompiling and re-playing the story. Figure 2 (b) shows that stories generally grew quickly at first, and then entered a period of fluctuating slow growth. 83% of story edits took place within the first 90 minutes (the duration of a block class period) of the story's creation, so most of the activity under investigation in this study took place within the classroom literacy space.




(a) Authorship distribution (b) Story growth over time

Figure 2. Stories written during the study.

This study's primary unit of analysis is the story edit, or differences between each pair of adjacent versions of a story. Comparing line-by-line changes in an edit yields a collection of ops, where each op is a contiguous set of lines inserted, deleted, or replaced from one version to the next. Each op was classified by the syntax elements

involved, such as knots or diverts. Some ops involve changes to code, such as creating new knots, diverts, and variables. Others only involve changes to text, leaving the story's programmatic structure unchanged. Then each edit was classified according to the properties of its ops and whether the prior and latter versions successfully compiled. Edits leaving the story unchanged were filtered out. Story edits inserted an average of 5.3 lines per edit ($\sigma$ =11.4) and deleted an average of 1.9 lines per edit ($\sigma$ = 4.2). Table 1 shows the classification scheme.

Table 1: Classification scheme for states of story edits

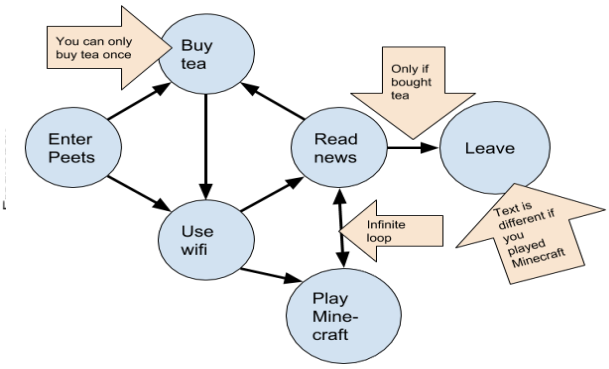
| Pre ok? | Post ok? | Ops | State | Simple state | Percent |
|---|---|---|---|---|---|
| yes | yes | >= 5 lines inserted total | MAJOR INSERT | OK | 14% |
| yes | yes | -5 < lines < 5; code changes | MINOR CODE EDIT | OK | 27% |
| yes | yes | -5 < lines < 5; no code changes | MINOR TEXT EDIT | OK | 22% |
| yes | yes | >= 5 lines deleted total | MAJOR DELETE | OK | 2% |
| no | yes | any | DEBUG SUCCESS | OK | 10% |
| yes | no | any | ERROR | ERROR | 11% |
| no | no | any | DEBUG FAIL | ERROR | 13% |

### Views of other stories

The second research question asks whether reading other stories was helpful in students' debugging. To answer this, backend log data was filtered to collect every instance of a student viewing a story. We then counted the number of a user's views which occurred during the span of each story edit (excluding views of the story being edited). Students viewed an average of 2.4 other stories ($\sigma$ = 2.6) during editing. This was surprising, as with only a few exceptions, students reported in the post survey that they enjoyed reading each others' stories and frequently cited specific examples of stories they had read. One possible explanation is that this study counts only views that took place during an edit.

### Summative assessment and survey

Two months after the interactive storytelling unit ended, participants were given a summative assessment of their debugging skills and a post-study survey collecting demographic information and measuring their affect and attitudes toward computing (Friend, 2016). Because participants had no prior experience with interactive storytelling, a pre/post test study design was not feasible. We used an adaptation of the Fairy performance assessment (Werner, et al., 2012), which was designed for this situation. Students were given a directed graph (Figure 3(b)) showing the desired functionality of a story (they had previously used directed graphs in planning their stories), annotated with specific issues. They were then given a copy of a broken implementation of the story and asked to fix it.




(a) Score distribution (b) Story graph showing desired functionality

Figure 3. In the summative assessment, students were asked to debug a broken story.

These assessments were scored using a rubric. One point was assigned for correctly linking story knots together using diverts (mapping the graph structure to the story structure), and one point was assigned for correcting each issue pointed out by an arrow. Each required the use of a different computational topic students had worked with during the unit: maintaining state explicitly with variables and implicitly with the players' path through the story, using state to change text output, and using state to control the availability of choices.

## Analysis

### RQ1: Does writing interactive stories support development of debugging practices?

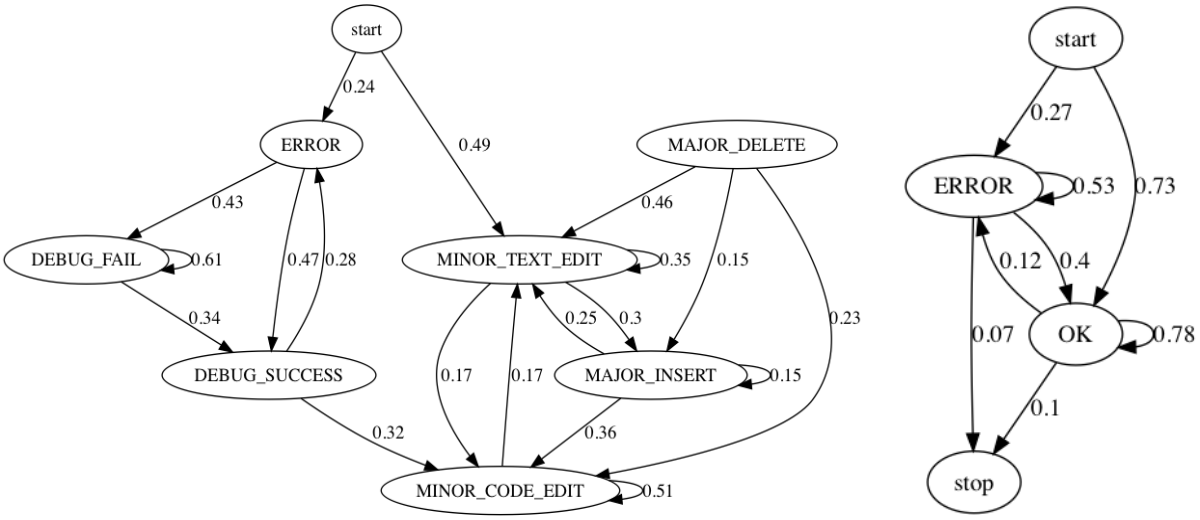
To answer this question, we first needed to describe patterns in students' story editing. We created a matrix counting the number of transitions from each pair of states, across all edits in all stories, and visualized this in a transition diagram. We also created a simplified model grouping together all successful edits and all unsuccessful edits. We then used OLS regression to estimate the association between writing more complex stories and summative score, and the association between probability of successful debugging and summative score. Story complexity was defined as the sum of diverts and choices used across all of an author's stories. This measure was chosen to capture the amount of computational content in stories. The probability of successful debugging was calculated as DEBUG_SUCCESS / (DEBUG_SUCCESS + DEBUG_FAIL) for each author.

### RQ2: Does reading other stories support development of debugging practices?

We theorized that as debugging is a distributed, social, and mediated process, students might be especially likely to view other stories (either written by peers or their own earlier stories) during debugging. As described in the previous section, we collected the number of other story views which occurred during each story edit. We grouped edits into debugging edits (DEBUG_SUCCESS and DEBUG_FAIL) and non-debugging edits (the rest), and conducted a two-tailed, two-sample independent T-test to determine whether a significant difference exists in these two groups' means, with the null hypothesis that there is no difference between group means. We do not assume equal sample size or variance between the groups, so we use Welch's T-test. Additionally, in the post-study survey, we asked students whether they found it helpful to look at other stories.

## Results

### RQ1: Does writing interactive stories support development of debugging practices?



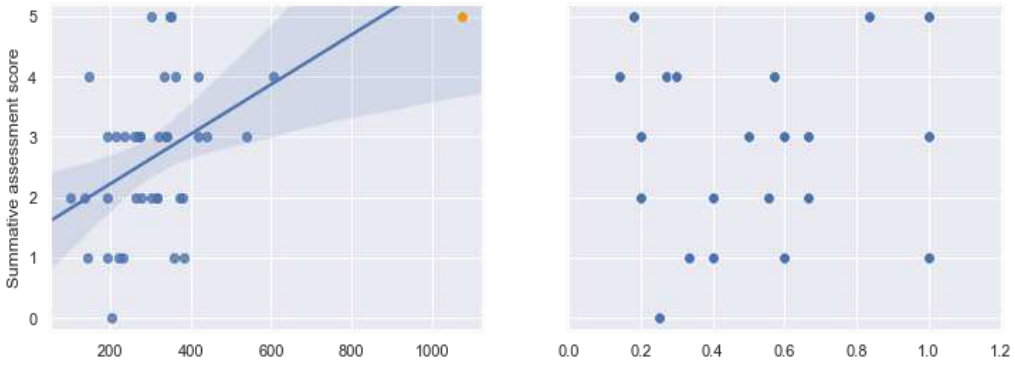

(a) Editing states (b) Simplified states

Figure 4. Transition diagrams showing probabilities of moving between story editing states.

Figure 4 shows the full (a) and simplified (b) transition diagrams between story editing states. (Transition probabilities under 0.15 are omitted for clarity.) Several editing patterns are visible: Authors tend to begin by entering one of two subgraphs: success or error (this was the inspiration for creating the simplified diagram in Figure 4(b)). When edits are successful, authors generally begin with short text editing (for example, writing a story with no code), and then cycle between editing text and code. MINOR_CODE_EDIT is a sink state, suggesting that once stories successfully mature, authors tend to spend their effort tweaking the code rather than adding substantial new textual content.

Of the successful editing states, DEBUG_SUCCESS stood out as most likely to precede an error. This is not surprising, as creating and fixing errors is part of the iterative process of writing challenging programs. When authors fail to successfully debug on encountering an error, they enter the DEBUG_FAIL sink state. The overall impression given by Figure 4(a), and clarified in Figure 4(b), is that of generally productive editing with distinct modes of successful editing and debugging.

Figure 5(a) shows a scatter plot and regression line (with 95% confidence interval) for the association between an author's total output of diverts and choices and score on the summative assessment. After removing the prolific outlier (shown in orange), the association was modeled as score = 1.58 + (diverts+choices) * 0.0035, with P>|t| = 0.004 and adjusted $r^2$= 0.18. In contrast to this strong association, there was no association between probability of successful debugging and summative score. This is clearly visible in Figure 5(b).

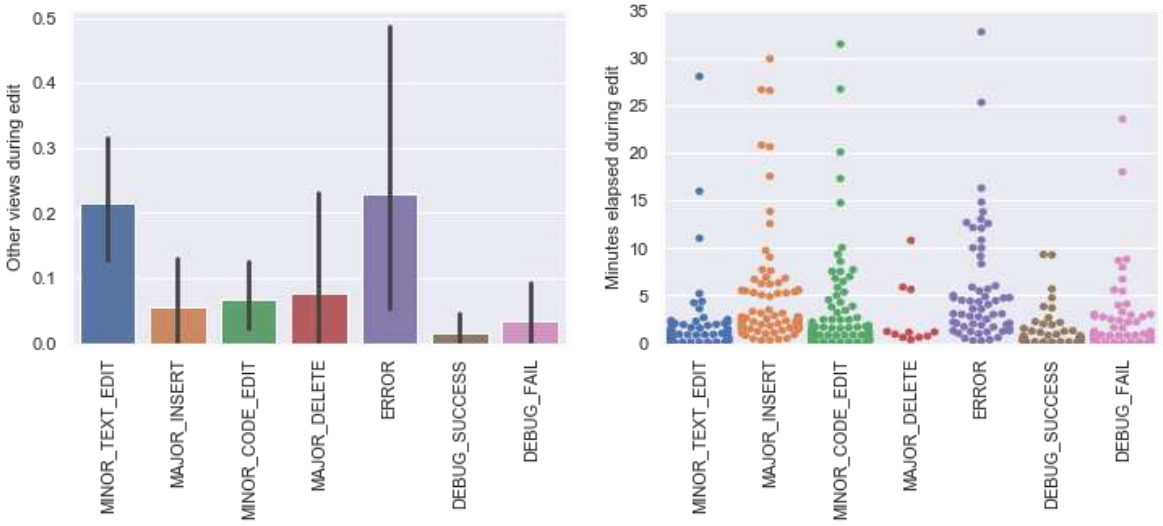



(a) Total diverts and choices (without outlier) (b) Probability of successful debugging

Figure 5. OLS Regression and 95% confidence interval plotted.

## RQ2: Does reading other stories support development of debugging practices?

Figure 6(a) shows the mean and standard deviation number of story views for each editing state. The result of the T-test comparing debugging states and others was t=-3.66, p >|t| = 0.0003. There was a significant difference between debugging and non-debugging states, but in the opposite direction as was hypothesized. Another unexpected difference was the dramatically higher number of story views during MINOR TEXT EDIT and ERROR states. We wondered whether this was due to these states tending to have longer time durations; Figure 6(b) plots the mean time interval for each editing state. We interpret these results in the next section.




(a) Mean views of other stories during edits (with σ) (a) Minutes elapsed during edits

Figure 6. Characteristics of different edit states.

## Discussion

This study's results suggest that a literacy-based approach to teaching computer science using interactive storytelling can be an effective context for learning debugging. The results also revealed surprising dynamics between debugging and viewing other stories, which warrant further iterations of design-based research. Participants in this study engaged in substantial debugging (moving in and out of error states) during their story-writing, and authors who used more computational elements in their stories were more likely to score highly on

the summative assessment. However, we did not find an association between successful debugging and higher performance on the summative task. These findings agree with participants' near-universal consensus that the literacy-based approach to teaching computer science helped them grow as programmers. Furthermore, when the post-study survey was administered, students had already spent two months learning Python, and 89% felt that interactive storytelling with Ink had helped prepare them to learn Python. (In contrast, transitioning from block-based languages to text-based languages is often difficult for novices (Weintrop & Wilensky, 2016).

One surprising finding was that, rather than using other stories as debugging resources, authors viewed stories less often during debugging. One interpretation of this is might be that debugging is a time to focus on the problem. Intuitively there is some truth to this, however this would also imply that looking at other stories during debugging should be negatively-correlated with debugging success. We did not find this to be the case. Furthermore, the literature on professional debugging (discussed above) finds a substantial reliance on other people and texts. It could be the case that other resources, such as handouts or the Ink language documentation, were more useful than other stories when authors got stuck.

The other unexpected pattern observed in story views across edit states was that authors looked at other stories so frequently while in ERROR and MINOR TEXT EDIT. The prevalence of other story views during ERROR may be partially explained by the fact that ERROR states tended to occupy somewhat more time than other states (see Figure 6(b)). Additionally, it is plausible that authors in ERROR edit states became frustrated and disengaged, and read other stories as a diversion rather than as part of productive debugging. The disproportionately high views of MINOR TEXT EDIT, however, cannot be explained by long duration; 90% of these edits spanned less than three minutes. The post-study survey provides a possible interpretation. All but three participants felt that reading peers' stories was helpful, but when asked to cite specific examples, students almost always described the content of the stories, and not their structural features. This, together with the high number of other views for the MINOR TEXT EDIT state, suggests that students may have been more successful learning from the content of each others' stories than from the computational aspects.

If this interpretation is correct, it poses a design challenge for future development: How might we design the literacy space so that students can learn from computational aspects of other stories as well as their content? One possible approach was tested at the end of the study, and has since been developed further. The students' final assignment was to collaboratively create a story in which the reader explores a world. The class planned out the story on one wall of the classroom, each student wrote a small part of the world, and then they worked together to import partial stories into a whole. Variables (such as energy level or the number of clues collected) maintained state across the different components. This assignment surfaced important computational concepts such as state and interface, and made students dependent on the computational aspects of each others' stories.

The results of this study should not be over-generalized. This study's participants attended an all-girls private school which provides unusual levels of personal attention (possibly allowing them to feel safe participating in a literacy space) and they had already studied computer science for one year using Scratch. This study's quantitative results have yet to be replicated with larger and more heterogeneous participants, and in-person ethnographic analysis of interactions within the literacy space is needed to validate our interpretations. This research is underway.

## Conclusion

At the conclusion of CSCL 2017, eight provocations were presented for the future of the field, including the question of whether the community ought focus on basic research and give up trying to make tangible change in the educational system (Wise & Schwartz, 2017). The prospect of widespread adoption of computer science and interdisciplinary computational thinking potentially means that the debate will shift from whether computers should be used in schools, to how. This ought to reinvigorate sociocultural research on how schools can support vibrant human-computer activity systems which attend to questions of culture, identity, power, and privilege, as well as developing students' technical skills. This paper contributes to the proposition that we do not have to choose between those goals.

## References


Beller, M., Spruit, N., & Zaidman, A. (n.d.). How developers debug.

Bonar, J., & Soloway, E. (1985). Preprogramming knowledge: A major source of misconceptions in novice programmers. Human–Computer Interaction, 1(2), 133–161.

Buechley, L. (2006). A Construction Kit for Electronic Textiles. In *2006 10th IEEE International Symposium on Wearable Computers* (pp. 83–90). Montreux, Switzerland: IEEE.

Burke, Q., & Kafai, Y. B. (2010). Programming & storytelling: opportunities for learning about coding & composition (p. 348). ACM Press.

Burke, Q., O'Byrne, W. I., & Kafai, Y. B. (2016). Computational Participation: Understanding Coding as an Extension of Literacy Instruction. Journal of Adolescent & Adult Literacy, 59(4), 371–375.
Denning, P. J. (2017). Remaining trouble spots with computational thinking. *Communications of the ACM*, *60*(6), 33–39.
Flood, V. J., DeLiema, D., Harrer, B. W., & Abrahamson, D. (2018). Enskilment in the Digital Age: The Interactional Work of Learning to Debug, 2.
Friend, M. (2016). *Women's History, Attitudes, and Experiences with Computing*. Dissertation.
Inkle. (2016). *Ink*. London, UK. Retrieved from https://github.com/inkle/ink
Ito, M. (2010). *Hanging out, messing around, and geeking out: Kids living and learning with new media*. Cambridge, MA: MIT Press.
K-12 Comptuer Science Framework. (2016).
Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. Communications of the ACM, 50(7), 58.
Kirschner, P. A., Sweller, J., & Clark, R. E. (2006). Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*, *41*(2), 75–86.
McCauley, R., Fitzgerald, S., Lewandowski, G., Murphy, L., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: a review of the literature from an educational perspective. Computer Science Education, 18(2), 67–92.
Pea, R. (1986). Language-independent conceptual bugs in novice programming. Journal of Educational Computing Research, 21, 25–36.
Piorkowski, D. J., Fleming, S. D., Kwan, I., Burnett, M. M., Scaffidi, C., Bellamy, R. K. E., & Jordahl, J. (2013). The whats and hows of programmers' foraging diets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13* (p. 3063). Paris, France: ACM Press.
Proctor, C., & Blikstein, P. (2019). Unfold Studio: Supporting critical literacies of text & code. Information and Learning Science, 2(1).
Proctor, C., & Garcia, A. (2019). Hogg, L., & Stockbridge, K. (Eds.). Student voices in the digital hubbub. *Giving student voice due weight: Possibilities and challenges in USA and New Zealand*.
Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … others. (2009). Scratch: programming for all. *Communications of the ACM*, *52*(11), 60–67.
Ware, P. D., & Warschauer, M. (2005). Hybrid literacy texts and practices in technology-intensive environments. International Journal of Educational Research, 43(7–8), 432–445.
Weaver, C. (1996). *Teaching Grammar in Context*. Portsmouth, NH: Boynton/Cook.
Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms. Journal of Science Education and Technology, 25(1), 127–147.
Weintrop, D., & Wilensky, U. (2016). Bringing Blocks-based Programming into High School Computer Science Classrooms, AERA.
Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (n.d.). The fairy performance assessment: measuring Cauleycomputational thinking in middle school, 6.
Wise, A. F., & Schwarz, B. B. (2017). Visions of CSCL: eight provocations for the future of the field. International Journal of Computer-Supported Collaborative Learning, 12(4), 423–467.