Story Programming: Explaining Computer Science Before Coding

Jennifer Parham-Mocello
Oregon State University, USA
parhammj@oregonstate.edu

Shannon Ernst
Oregon State University, USA
ernstsh@oregonstate.edu

Martin Erwig
Oregon State University, USA
erwig@oregonstate.edu

Lily Shellhammer
Oregon State University, USA
shellhal@oregonstate.edu

Emily Dominguez
Oregon State University, USA
domingue@oregonstate.edu

ABSTRACT

Story Programming is an approach for teaching complex computational and algorithmic thinking skills using simple stories anyone can relate to. One could learn these skills independent of a computer or with the use of a computer as a tool to interact with the computation in the tale. This research study examines the use of Story Programming before teaching coding in a computer science orientation course to determine if it is a viable alternative to the code-focused way of teaching the class in the past. We measure the viability of the Story Programming approach by evaluating student-success and learning outcomes, as well as student reactions to post-survey questions.

CCS CONCEPTS

• Social and professional topics ~ Computational thinking • Social and professional topics ~ Computer science education • Applied computing ~ Interactive learning environments

KEYWORDS

Story Telling; Computer Science Pedagogy; Introduction to Computer Science

ACM Reference format:

Jennifer Parham-Mocello, Shannon Ernst, Martin Erwig, Lily Shellhammer, and Emily Dominguez. 2019. Story Programming: Explaining Computer Science Before Coding. In Proc. of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), Feb. 27-March 2, 2019, Minneapolis, MN, USA. ACM, *NY, NY, USA, 7 pages.* https://doi.org/10.1145/3287324.3287397

1 INTRODUCTION

Many approaches to introducing computer science to students are predicated on programming, that is, they require an understanding of how to code an algorithm in a programming language. This approach is used in efforts such as *code.org* that promote coding and computer science to younger children, but the abstract nature of programming languages (block based or not) can pose a significant barrier to entry.

* This work is partially supported by the National Science Foundation under the grant CCF-1717300.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02…$15.00
https://doi.org/10.1145/3287324.3287397

A code-first approach can be effective when students have a good understanding of programming or are willing to acquire it. Since computer science is not synonymous with programming, there is no inherent necessity to tie the orientation to computer science to coding activities. Students who are not sure whether they want to study computer science but are curious about the subject should not be excluded because they are reluctant to the idea of having to become a programmer as a prerequisite to understanding computer science. The same applies to laypeople who want to get some basic understanding of a field of growing importance for society. Therefore, efforts to explain computer science without a computer, such as *csunplugged.org*, have gained popularity, especially among the K-12 community, and studies show that this approach broadens participation [3].

The researchers in this study believe that the state-of-the-art introductory computer science education at the university level could benefit from more creativity and computational thinking without the use of a computer. To reach a wide audience with a diverse background and set of expectations, we use an approach that uses well-known stories and everyday situations to explain computer science concepts before teaching coding [8]. Explaining computational concepts through popular stories has three complementary advantages.

First, identifying computing concepts in stories and everyday situations shows that computation is a universal phenomenon that does not only occur in machines. Pointing out the occurrence of computing in everyday situations emphasizes the relevance of computing and provides motivation for understanding basic concepts of computer science independent of the goal to become a programmer or computer scientist.

Second, the use of well-known stories can help with making the learning curve gentler, since students only need to understand the link between the story elements and the computing concepts. If they know the story, the objects and events are readily available as building blocks for computing metaphors. This is different in approaches that invent new stories to explain computing such as [2, 14] where students first have to absorb and understand the story and only then can process the links to computing concepts.

Third, stories can make people empathize more with the problems. Considering the problem of finding the shortest path from the couch to the fridge may be important, but it does not reach the level of importance of the path-finding problem that Hansel and Gretel face. Being emotionally engaged in a problem often means to care more about a problem, which helps to make the explanation provided by the story more memorable and effective. This aspect is probably stronger

by using existing popular stories whose impact on the audience has already been demonstrated through their popularity.

In the remainder of this paper we report on a study that researches an approach called *Story Programming* as an alternative for teaching a computer science orientation course, CS 0. This approach is based on the book *Once Upon an Algorithm: How Stories Explain Computing* [7], and this research investigates whether this approach is a viable option for teaching a university-level computer science course. We define viable in terms of satisfying learning outcomes and positive student reactions to post-survey question.

2 BACKGROUND

The two most closely related areas to our approach are the so-called "unplugged" computational thinking approach and the other existing approaches to using stories for explaining computing concepts.

2.1 Unplugged Computational Thinking Activities

The most well-known approach to teaching computer science concepts without the use of a computer is the approach taken in *csunplugged.org* [4]. It is a collection of engaging activities that can illustrate computing concepts, but it does not use stories. Story Programming uses the idea of unplugged activities performed without a computer, but the actual activities in this project are different and relate to the stories in the book or stories students create.

Unplugged activities are primarily used in K-12 [1, 5, 11, 17, 18], but this research study employs the idea of teaching computational thinking without a computer at the university level. Most alternatives for teaching introductory computer science courses in institutions focus on changing the curriculum in their introductory computer science classes to improve success and retention [9, 16] and make topics covered more relevant and broader [15]. Some institutions have created interest-based classes allowing students to choose a class section based on what they like, such as game development, robotics, music, and mobile applications [9, 20], while others focus on adding computational thinking to their curricula with and without the use of a computer [10, 13, 16, 19]. These studies show that teaching computational thinking helps students think about different ways to attack problems, making them more effective problem solvers. This study uses a Story Programming approach to teach computational thinking skills using stories without a computer before teaching programming skills.

2.2 Story Programming Approaches

The use of stories to explain computing is not new. *Computational Fairy Tales* describes algorithms and data structures as part of a story about a princess on a quest to save her father's kingdom [14]. The target audience is middle school children, and the treatment of concepts is often quite brief. The selection of topics is ad hoc, and the book does not cover any language aspects. *Lauren Ipsum* [2] employs a similar approach. It tells a story about a girl who gets lost in a forest and wants to find her way back home. In her adventure, she has to solve several problems, which serve as a hook to introduce concepts of algorithms and math on a very high level. The target audience is also middle school children, and the story is like *Alice in Wonderland* with its playful and clever use of names. It contains an appendix that provides additional explanations of the concepts mentioned in the story, but it also does not cover any language aspects.

One study used *Computational Fairy Tales* to help the retention and academic performance of computer science majors, mostly aimed at students with little to no programming experience [16]. It found that "CS0 students without prior programming experience got significantly higher grades in CS1 than CS0 students who had programmed before"; the students were split on how useful the book was to their learning. This is different than the study presented in this paper, which determines if Story Programming is a viable alternative to the traditional programming-focused approach for teaching a computer science orientation class. Another study used physical simulations in class to explain concepts, which students said helped them to understand the computational concepts [13], arguing that computer science or programming concepts could be explained effectively using stories if the connection between concept and story is strong. One study claimed that using a story to learn a concept will be easily accessible because that is how many of us learn to begin with [12]. These claims align with the rationale for using Story Programming, but the study presented in this paper does not investigate these claims. One other study used "unplugged" activities and storytelling to introduce teachers to computational thinking, but it focused on teacher training and used contextual stories to relate different "unplugged" activities to specific computational skills for teachers as the storytelling approach [6].

3 RESEARCH METHOD

At Oregon State University, students in the College of Engineering are required to take an orientation course to fulfill a degree requirement, and Computer Science Orientation (CS 0) is offered once a year to fulfill that requirement for students interested in majoring in CS. This course is primarily taken by incoming first-year students who are declared Computer Science majors, but students with prior computer science experience or outside the major may take the course as well. In the past, Python was used as the coding language with students beginning to write small programs as early as week two in a ten-week term. The lectures are focused on teaching basic Python including variables, control flow (both conditional statements and looping), functions and lists with exposure to how to design solutions to computer science problems and the idea of testing. In-class exercises are completed in groups and used to stimulate learning in a computer-free environment.

We divided the Fall 2017 CS 0 course into three sections. One section remained taught in the traditional fashion with a focus on programming in Python. The other two sections were taught using a new approach supported by *Once Upon an Algorithm*. One Story Programming section was taught using Python in the second half of the class, and the other section used Haskell to investigate differences in language choice with the new approach. All sections were taught by the same instructor, and the students were placed in sections at random.

3.1 Course Structure

All three sections had about 100 students per lecture and utilized presentation slides to teach concepts, as well as live coding

demonstrations through a terminal when programming was being taught. Every week students engaged in in-lecture, group exercises. While the traditional section primarily focused on programming concepts, these activities did not involve a computer. They often centered around writing pseudocode, designing a solution to a problem, or analyzing code. The Story Programming in-class exercises did not focus on pseudocode or code analysis until half way through the term, and most of their exercises focused on understanding the stories used in the book *Once Upon an Algorithm* from a computational perspective, coming up with new stories to explain computational concepts, and developing and tracing algorithms. Since these activities did not use a computer, they were referred to as computational thinking (CT) activities rather than coding activities.

The students in the Story Programming sections were required to read relevant chapters of the book before class with a weekly online quiz. Chapters were generally covered in a sequential fashion with some occasional skipping to group like concepts together. The traditional section did not have a textbook, but students were encouraged to use online documentation for Python to supplement their learning where needed. The use of the book in the Story Programming sections changed the emphasis placed on some of the concepts, allowing for greater breadth and depth of concepts. For example, the Story Programming sections covered the concept of data structures and decidability more deeply than the traditional section, which gave a cursory view of data structures only through lists and only mentioned the idea that not all problems are solvable.

Each section had one two-hour lab per week, but while the traditional section labs focused on programming activities beginning in week 1, the first 5 Story Programming labs focused on small group activities applying concepts to the real world. For example, one of the first activities examined path finding algorithms using the tale of Hansel and Gretel, and students were presented with three variants of the algorithm that they had to act out with pebbles. Another activity helped students learn about the runtime of different algorithms by having to count and transfer beans across the classroom using different methods. The programming activities in the last 5 labs mirrored in code the concepts that were covered in earlier labs, rather than relating to the new concepts.

Each section had weekly assignments. For the traditional section these focused on programming. The students were presented with a problem statement and were asked to implement a solution in Python by the end of the week. These programs often focused on interacting with a user. The Story Programming sections focused on describing algorithms and connecting concepts they were learning from the book to the real world. When the students in the Story Programming sections began programming, their programs were much smaller toy problems and did not involve user input. The problems given to the Story Programming sections were also slightly different depending on the programming language.

3.2 Research Questions

With clearly defined sections, a formal study on the differences in pedagogy can address the following broad research questions, which we will outline in more detail in section 4.

RQ1: Does the Story Programming approach satisfy the learning outcomes?
RQ2: How do students react to the Story Programming approach?
RQ3: How do students react toward the CT activities versus coding activities?

3.3 Data Collection

With IRB permission, course-level DWF rates and grade distribution information were collected from the registrar, and student-level post-survey data were collected from consenting participants. Conceptual questions about algorithms, representation, and abstraction on a post-survey addressed the first research question, and survey questions with a 4-point Likert scale addressed the other two research questions.

4 RESULTS AND DISCUSSION

The course-level student success results are out of 277 students, and the student-level post-survey results are out of 110 consenting participants (35 Story Python, 25 Story Haskell, and 50 Traditional). Since the data are not normally distributed and are based on a Likert scale, non-parametric statistical tests, such as Kendall's correlation tau, τ, and the Kruskal-Wallis hypothesis test are used to reject hypotheses with 95% confidence, α=.05.

RQ1: Does the Story Programming approach satisfy the learning outcomes?

A college-level learning outcome for this class is student success. Two measures of student success are DWF rates and the grade distribution of passing students. A course-level learning outcome in CS 0 is to gain an understanding of computer science, which is measured using a post-survey.

DWF Rates

The results in Table 1 do not show a significant difference in the DWF rates between the two Story Programming sections or between the Story Programming and traditional approaches.

Sections	Students and DWF rates	
	# Students	*DWF*
Story Python	105	8.6%
Story Haskell	65	9.2%
Traditional	107	6.5%

Table 1: Number of students and DWF in each section

This suggests that the Story Programming and traditional approaches have comparable student retention and performance, and the choice of language in the Story Programming approach does not affect this. As a side note, the average DWF rate across all sections is almost 11% lower than the previous year, when all students were in one section of 284. This suggests that class size has a bigger effect on retention and student success than the approach or language used in a computer science orientation class.

Grade Distribution

Statistical tests show no differences in grade distributions between the Story Programming populations using different languages or between the two different approaches. However, it is interesting to note that the

Story Programming sections differ the most (see Figure 1). The Haskell section has a lower percentage of As and a higher percentage of Bs.

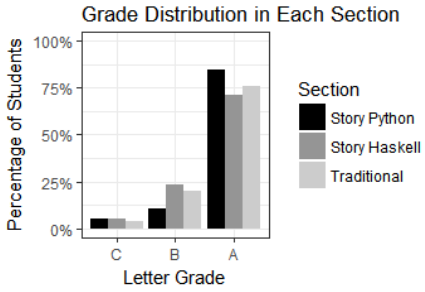

Figure 1: Grade Distribution among Story Programming approaches and the traditional programming approach.

Post-Survey Conceptual Questions

Out of the 110 consenting participants across all sections, there are no differences in the way students from the Story Programming sections or different approaches answer the conceptual post-survey questions (see Table 2). Interestingly, approximately half the students answer the algorithm and abstraction questions correctly, whereas most students answer the representation questions incorrectly.

Sections	Post-Survey Conceptual Questions		
	Algorithm (8)	*Representation (4)*	*Abstraction (4)*
Story Python	47.9%	42.1%	60.0%
Story Haskell	54.0%	35.0%	63.0%
Traditional	52.8%	36.5%	61.0%

Table 2: Percentage of students with Correct Answers to Post-Survey Conceptual Questions

RQ2: How do students react to the Story Programming approach?

Only students from the two Story Programming sections answer survey questions about the new approach and the use of the book. The following questions are answered by comparing these two sections,

Does the chosen language make a difference in how students feel about the Story Programming approach?

There is a significant difference in the student reactions toward the Story Programming approach based on the section (*pvalue=.003*). Figure 2 shows the distributions of student feelings about the approach in the two sections. The Story Programming section using Haskell tends to like the approach more than the section using Python. However, an average of 65% of the students in both sections really or somewhat like the Story Programming approach to teaching the orientation class with only 13% disliking the approach a great deal.

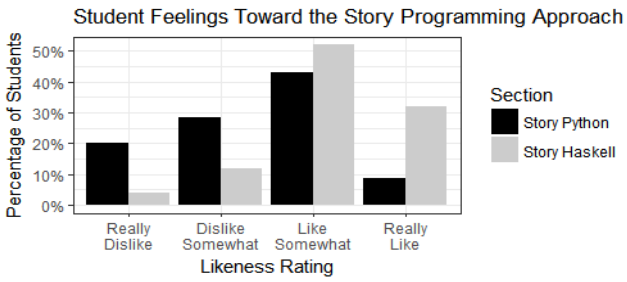

Figure 2: Distribution of students' feelings about the Story Programming approach in the different sections.

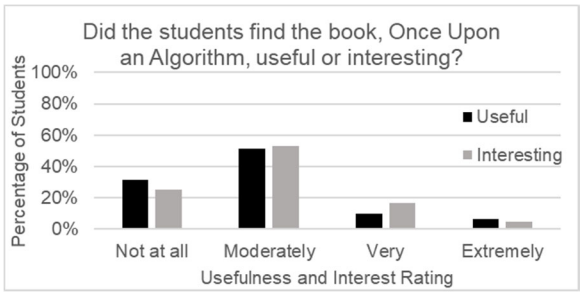

Figure 3: Percent of students who find the textbook useful to their learning or interesting.

Does the chosen language make a difference in the way students feel about the book used?

The two sections do not significantly differ in the responses regarding usefulness of and interest in the book, which means that students in both sections are consistent in the way they evaluate the book. This is observed in the strong positive, τ=*0.70*, correlation between student ratings on how useful the book is to their learning and how interesting it is, and there is a moderately positive correlation between how they feel about the Story Programming approach and whether they find the book useful (τ=*0.59*) or interesting (τ=*0.52*). Only 25-32% from both sections do not at all find the book useful to their learning or interesting, but most only find the book moderately useful or interesting (see Figure 5).

Figure 4 shows the students' favorite chapters are 1 (about algorithms and computation), 4 (data structures), and 6 (sorting). Since each of these chapters use different stories, this means that students are not partial to one specific story. Moreover, no student seemed to like chapter 15 (about abstraction), which uses Harry Potter as the story. Interestingly, the two Story Programming sections did not differ significantly in the choice of favorite chapters, but they did differ significantly in their least favorite chapters (*pvalue=.02*). The least favorite chapters are more evenly distributed with no one disliking chapter 2 (runtime and resources). It is interesting to note that the students tend to like the first part of the book more, which is about algorithmic concepts, in contrast to the later chapters with more abstract content about language concepts.

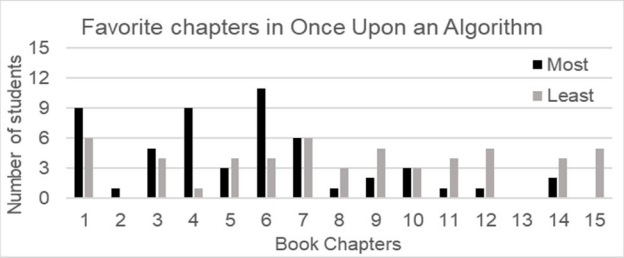

Figure 4: Number of students who choose specific chapters in the textbook as their most and least favorite.

Does prior programming experience, gender, or class standing change the way students feel about the approach?

Figure 5 shows that many students enter CS 0 with prior programming experience, and there is not a significant difference between sections. Since approximately 60-70% of the students have prior programming experience, this is a variable worth considering.

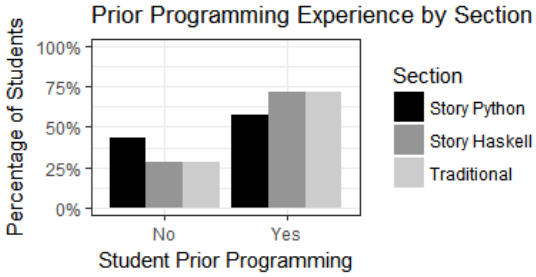

Figure 5: Prior programming experience across all three sections.

The data from the two Story Programming sections do not show a significant difference in the way students felt about the approach based on prior programming experience, and there is no strong correlation between prior programming experience and their feelings toward the Story Programming approach. However, it is interesting to note that there is a very low negative correlation, which means that there were a few more students with prior programming experience who have negative feelings toward the approach. There is also no difference in the usefulness and interest ratings of the book among those with and without prior programming experience.

There is no difference in gender or class standing distributions in the two Story Programming sections, and there is not a significant difference in class standing across all sections. However, Figure 6 shows a significant difference in gender across the Story Programming and traditional sections (*pvalue=.04*). The traditional section has more male students.

Figure 6: Gender demographics across all three sections.

Just as with prior programming experience, there is not a significant difference in the way students felt about the approach or textbook based on gender or class standing, and there is not a strong correlation between gender or class standing with how they felt about the approach or book.

RQ3: How do students react toward the CT activities versus coding activities?

There is a moderate to strong positive correlation, τ*=0.57-0.72*, between how students feel about the CT activities helping them learn, whether they feel the CT activities motivate them to learn more about computer science, and how engaging they rate the CT activities. There is a moderate correlation between how a student feels about the Story Programming approach with how they rate the CT activities. The students who like the approach are more likely to find the activities engaging (τ*=0.57*) or feel that they help them learn (τ*=0.49*). Likewise, the rating for the coding activities show similar correlations, τ*=0.58-0.72*, between how students feel about the coding activities helping them learn, whether they feel the coding activities motivate them to learn more about computer science, and how engaging they rate the coding activities.

Do the approaches lead to differences in the activities students consider helpful for learning, motivating, and engaging?

There is a significant difference in student reactions toward CT activities among the Story Programming sections, but there is not a significant difference in student reactions toward CT activities in the sections using different approaches. Students in the two Story Programming sections differ on whether they feel CT activities help them learn (*pvalue=.03*), motivate them to learn more about computer science (*pvalue=.01*), and engage them (*pvalue=.01*). Since the trends are the same for these students feelings toward CT activities, we include one figure showing the differences in motivation to learn more about CS among the various sections. Figure 7 shows that the students in the Haskell Story Programming section strongly agree more than those in the Python section, as well as disagreeing less.

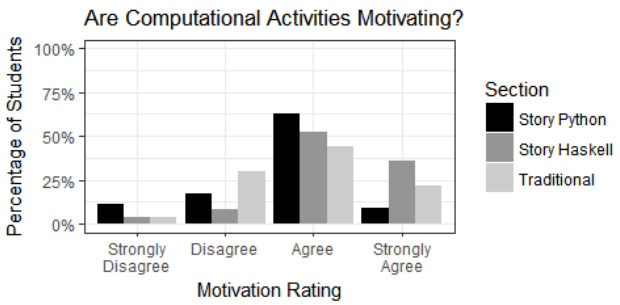

Figure 7: Distribution of agreement about whether computational thinking activities motivated students' learning.

There is not a significant difference in student reactions toward coding activities among Story Programming sections or across the different approaches. The Story Programming sections have about 10% more students who feel the coding activities help them learn, motivate them, and are engaging, but the traditional section has 24% more students who feel like the coding activities motivate them over the CT activities. This might be due to a heavier focus on writing code in the traditional approach, whereas the Story Programming approach focuses on CT using stories and non-coding activities for the first half of the term. Overall, the students in the traditional section disagree more about the helpfulness, motivation, and engagement of both kinds of activities, but the students in the Story Programming section with Python strongly disagree more than any other section.

Is there a correlation between the way students rate CT activities versus coding activities?

Most students who agree the CT activities are engaging also agree the coding activities are engaging, and this is the same for those who strongly agree. Even though there is not a strong linear relationship between most student ratings for CT activities and coding activities, there are other interesting patterns to observe. Across all sections, there are no students who strongly agree with the CT activities and then disagree or strongly disagree with the coding activities. Interestingly, the Haskell Story Programming section never disagrees or strongly disagrees to either activity. In all sections, if students disagree with CT activities, then they tend to agree with coding activities. However, if they disagree with the coding activities, then they continue to disagree with the CT activities.

None of the sections show a difference in the way students of different genders rate the CT or coding activities. Across all sections, there is a significant difference in the way students of different genders rate the engagement of coding activities (*pvalue=.01*), and female students tend to strongly agree with the engagement of these activities more than their male peers. However, students with different prior programming experiences in the Python Story Programming section differ in all their ratings of CT activities (*pvalues=.01-.05*), but ratings for the coding activities in this section do not differ based on prior programming. No other sections show differences in their ratings of CT or coding activities based on prior programming, and across all sections, there is a significant difference among students with prior programming experience and their ratings for the motivation and engagement of CT activities (*pvalues=.03 and .04*).

5 CONCLUSIONS

This study does not show any difference in student success or conceptual learning among either Story Programming sections or between the two approaches. However, it is interesting that the choice of Haskell provides a modest difference in the distribution of As and Bs. It appears that the approach or language choice across different sections of an orientation course does not impact DWF rates as much as having smaller sections does. This result supports making class sizes smaller and perhaps providing different choices for students when offering multiple sections of a course.

Student reactions to the Story Programming approach are mostly positive, and students' prior programming experience, gender, or class standing does not seem to impact the way students react to the new approach or the book. However, the choice of Haskell with the Story Programming approach seems to create more positive reactions than Python. In general, students find the book to be moderately useful or interesting. The students like the chapters in the beginning of the book the most, which suggests that students in an orientation course like concrete algorithmic concepts more than abstract computational concepts.

Interestingly, the two approaches show no significant difference in student reactions toward CT activities performed without the use of a computer and coding activities using a computer, but the language used with the Story Programming approach may affect reactions toward the CT activities. Students in the Haskell section strongly agree more than the Python section with the positive impact of the CT activities, but the reactions toward the coding activities in these sections do not differ. In all sections, students rank coding activities higher than CT activities, which suggests that students value coding more than the CT activities.

In summary, we conclude that Story Programming is a viable approach for teaching a CS orientation class with a mixture of students who do and do not have prior programming experience. The choice of language used with the Story Programming approach has some impact on student grades and reactions toward the approach and activities.

REFERENCES

[1] T. Bell, J. Alexander, I. Freeman, and M. Grimley. 2009. Computer Science Unplugged: school students doing real computing without computers. Journal of Applied Computing and Information Technology 13, 1.

[2] Carlos Bueno. 2014. Loren Ipsum. No Starch Press.

[3] Thomas J. Cortina. 2015. Reaching a broader population of students through "unplugged" activities. Commun. ACM 58, 3 (February 2015), 25-27. DOI: https://doi.org/10.1145/2723671

[4] CS Education Research Group. CS unplugged: Computer Science without a computer. http://www.csunplugged.org

[5] Paul Curzon. 2013. cs4fn and computational thinking unplugged. In Proceedings of the 8th Workshop in Primary and Secondary Computing Education (WiPSE '13). ACM, New York, NY, USA, 47-50. DOI: http://doi.acm.org/10.1145/2532748.2611263

[6] Paul Curzon, Peter W. McOwan, Nicola Plant, and Laura R. Meagher. 2014. Introducing teachers to computational thinking using unplugged storytelling. In Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE '14). ACM, New York, NY, USA, 89-92. DOI=http://dx.doi.org/10.1145/2670757.2670767.

[7] Martin Erwig. 2017. Once Upon an Algorithm: How Stories Explain Computing. MIT Press.

[8] Martin Erwig. 2017. The Real Ghost in the Machine. The World Today, 36-37 (Oct./Nov. 2017).

[9] Michael Haungs, Christopher Clark, John Clements, and David Janzen. 2012. Improving first-year success and retention through interest-based CS0 courses. In Proceedings of the 43rd ACM technical symposium on Computer Science Education (SIGCSE '12). ACM, New York, NY, USA, 589-594. DOI=http://dx.doi.org/10.1145/2157136.2157307

[10] Peter B. Henderson. 2011. Computing unplugged enrichment. ACM Inroads 2, 3 (August 2011), 24-25. DOI=http://dx.doi.org/10.1145/ 2003616.2003626

[11] Felienne Hermans and Efthimia Aivaloglou. 2017. To Scratch or not to Scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons. In Proceedings of the 12th Workshop on Primary and Secondary Computing Education (WiPSCE '17), Erik Barendsen and Peter Hubwieser (Eds.). ACM, New York, NY, USA, 49-56. DOI: https://doi.org/10.1145/3137065.3137072

[12] William J. Joel. 2013. A story paradigm for computer science education. In Proceedings of the 18th ACM conference on Innovation and technology in computer science education (ITiCSE '13). ACM, New York, NY, USA, 362-362. DOI=http://dx.doi.org/10.1145/2462476.2466526

[13] Dennis Kafura and Deborah Tatar. 2011. Initial experience with a computational thinking course for computer science students. In Proceedings of the 42nd ACM technical symposium on Computer science education (SIGCSE '11). ACM, New York, NY, USA, 251-256. DOI=http://dx.doi.org/10.1145/1953163.1953242

[14] Jeremy Kubica. 2012. Computational Fairy Tales. CreateSpace Independent Publishing Platform

[15] David J. Malan. 2010. Reinventing CS50. In Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10). ACM, New York, NY, USA, 152-156. DOI: http://dx.doi.org/10.1145/ 1734263.1734316

[16] Cindy Marling and David Juedes. 2016. CS0 for Computer Science Majors at Ohio University. In Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16). ACM, New York, NY, USA, 138-143. DOI: https://doi.org/10.1145/2839509.2844624

[17] Brandon Rodriguez, Cyndi Rader, and Tracy Camp. 2016. Using Student Performance to Assess CS Unplugged Activities in a Classroom Environment. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16). ACM, New York, NY, USA, 95-100. DOI: https://doi.org/10.1145/2899415.2899465

[18] Renate Thies and Jan Vahrenhold. 2013. On plugging "unplugged" into CS classes. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 365-370. DOI: http://dx.doi.org/10.1145/2445196.2445303

[19] Michele Van Dyne and Jeffrey Braun. 2014. Effectiveness of a computational thinking (CS0) course on student analytical skills. In Proceedings of the 45th ACM technical symposium on Computer science education (SIGCSE '14). ACM, New York, NY, USA, 133-138. DOI: http://dx.doi.org/10.1145/ 2538862.2538956

[20] Zoë J. Wood, John Clements, Zachary Peterson, David Janzen, Hugh Smith, Michael Haungs, Julie Workman, John Bellardo, and Bruce DeBruhl. 2018. Mixed Approaches to CS0: Exploring Topic and Pedagogy Variance after Six Years of CS0. In Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). ACM, New York, NY, USA, 20-25. DOI: https://doi.org/10.1145/3159450.3159592.