Bootstrap: Going Beyond Programming in After-School Computer Science

Emmanuel Schanzer
Harvard University Graduate School of Education
Boston, MA, USA
schanzer@bootstrapworld.org

Kathi Fisler
WPI Dept of Computer Science
Worcester, MA
USA
kfisler@cs.wpi.edu

Shriram Krishnamurthi
Brown University Computer Science
Providence, RI
USA
sk@cs.brown.edu

ABSTRACT

Adding computer science to already-packed middle- and high-school curricula can be difficult; after-school programs offer an enticing alternative to broadening student exposure to computing. Over the last eight years, we have deployed a content-rich introductory computing course to over a thousand middle-school students through after-school programs nationwide. Our program, Bootstrap, teaches students to program their own videogames in a way that connects deeply to in-school learning goals for algebra and coordinate geometry. Volunteers (college students or software professionals) teach Bootstrap through established after-school partners. This paper describes both Bootstrap and lessons we have learned about teaching computing effectively in after-school programs using volunteer teachers.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Curriculum, Computer science education.

General Terms

Design, Languages.

Keywords

After-school, graphical programming, animation, videogames, design-driven programming, middle-school, high-school

1. INTRODUCTION

Several countries, including the United States, have efforts to expand computing education in the pre-college years. These have led to initiatives to train significant numbers of computing teachers and develop curricula suitable for pre-college students. These efforts dovetail with more general efforts to prepare more students for education and careers in STEM (Science, Technology, Engineering and Mathematics).

Research suggests that STEM education is particularly important in the 10-13 age range (middle- to early high-school in the USA), especially for student groups who are under-represented in STEM. Students in this age range have the cognitive and intellectual maturity to approach more abstract material (hence the introduction of algebra at this age) [5]. This is also a period in which students start to make broad decisions about their career paths: they may decide whether they are "good" in math, which is a gatekeeper to most other classes that prepare students for STEM careers. Exposing students to computing in ways that builds their confidence in STEM could enable many more students to pursue computing and STEM courses or careers.

Creating solid computing experiences for under-represented 10-13 year olds (henceforth "tweens") is a daunting task. Detailed curricula and teachers with computer science background are scarce, particularly pre high-school. Even if schools in under-represented areas boasted adequate computing teachers, schedules are already packed with material and learning goals; in the USA, curricula are constrained by high-stakes tests in accordance with national initiatives such as No Child Left Behind. This suggests that near-term efforts should look beyond formal classrooms to engage under-represented students in computing.

This paper describes Bootstrap, a computing curriculum and after-school program for tweens with several notable features:

- It has clear learning objectives; it does not only aim to engage and excite students about computing.
- It uses videogame programming to reinforce algebra and coordinate geometry in hopes of improving students' experience with and confidence in math. The game structure connects specific mathematical concepts to concrete game behaviors. The connections to in-school math courses are clearly defined relative to USA standards for middle-school mathematics.
- It uses volunteers from IT organizations and universities to deliver classes through networks of established after-school programs. The programs provide the logistics (e.g., space, computers, student recruitment, legal protections); Bootstrap recruits, trains, and supports the volunteers.

This design attempts to tap into a variety of needs: focusing on after-school routes around scarcity of computing courses and teachers in under-represented areas; recruiting volunteers gives a structured outlet for college students and IT professionals who wish to teach programming as community service; combining videogames, computing and math education motivates and helps students who feel uncomfortable in math; providing content-rich STEM programs helps ambitious after-school organizations diversify their offerings beyond the arts and professions (such as finance, law, and journalism).

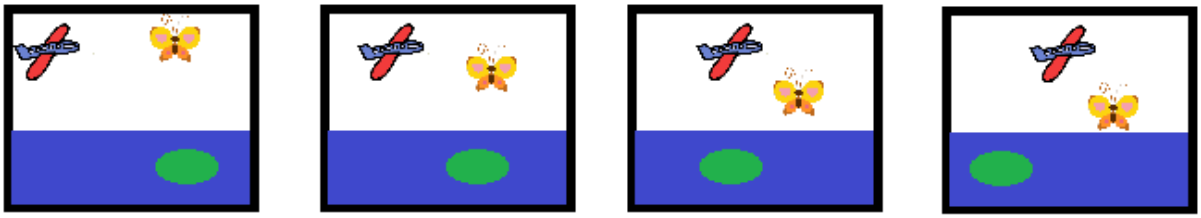

Figure 1: Sequence of Frames Making Up an Animation

Our Videogame

Created by (write your names): _____

Background

Our game takes place _____.
(In space? The desert? A mall?)

The Player

The player is a _____.

The player moves only up and down.

The Target

Your player GAINS points when they hit the target.

The Target is a _____.

The Target moves only to the left and right.

The Danger

Your player LOSES points when they hit the danger.

The Danger is a _____.

The Danger moves only to the left and right.

Figure 2: A Game-Design Worksheet

Since 2005, we have taught Bootstrap to roughly 1400 at-risk and under-represented students across the USA, through organizations with limited resources and obsolete computer labs. We have learned a lot about how to design volunteer-based after-school programs to teach computing in such environments. This paper describes our program design in detail and reflects on what we have learned. Our goal is to provide useful insights for other projects seeking to work through after-school programs to provide serious computing education to under-represented students.

2. THE CURRICULAR DESIGN

Bootstrap comprises ten 90-minute classes, the last of which is a showcase in which students present their games to friends, family, and teachers. We assume that students have used computers before for routine tasks such as web-browsing, but do not have prior programming experience. We assume that students are familiar with arithmetic and may have begun seeing functions and word problems as well, even if they may not be comfortable with this material. We also assume that students have played videogames and are drawn to the idea of building one. At the end of the course, each student (or pair of students) should have implemented a game of her own design that involves side-scrolling movement (meaning when characters scroll off one edge of the screen, they re-appear on the other side), user-input via key presses, and scoring based on collision of two game elements.

Our curricular design and its connections to math education build on a particular view of what a videogame is and how it is constructed. We now present this view in the same manner that we present it to students on the first day of a Bootstrap class.

2.1 Dissecting Games

Figure 1 shows four frames from a sample videogame in which the player moves the butterfly up and down attempting to land on the lily pad while avoiding the airplane. To understand how this game is built, we first ask students to identify what changes and how across successive frames. Students fill in a *game-dissection worksheet* with the details. The worksheet for this example says:

Thing in the Game	What changes about it?	More specifically …
Plane	position	x-coordinate
Butterfly	position	y-coordinate
LilyPad	position	x-coordinate
Water	nothing	nothing

A game-dissection worksheet

Understanding what does not move is important: visual elements that do not move are part of the background image for the game. These elements are fixed throughout gameplay. We will treat them differently from elements that do move when designing and implementing a game. Attributes other than position, such as size or color, could also change during a game.

The game-dissection table illustrates the first connection between Bootstrap and middle-school mathematics: the idea and role of coordinates arises early in the first day. Bootstrap does not assume students already know this material (in practice some do, some don't, and some have seen but not understood coordinates). Our lesson plans include several exercises on coordinates, grounded in the context of videogame screenshots.

Once students see that a game is made up of a collection of objects that change attributes from frame to frame and some fixed background information, they are ready to outline their own games. The Bootstrap curriculum requires that each game have three elements: a player, a target, and a danger. Each element can move in one dimension. The player attempts to collide with the target while avoiding collision with the danger; each collision with the target increases the player's score, while each collision with the danger decreases the score. Students are free to choose what those elements represent and the setting in which their game takes place. Students complete the *Game Design Worksheet* in Figure 2 to summarize their choices. Student designs vary widely, with sample games being about catching candy, avoiding monsters, flying through space, or collecting coins.

Table 1: Learning and Project goals for each lesson: learning goals target particular programming and mathematical concepts, while project goals enhance behavior of the game.

1. Lesson	2. Learning Goals	3. Project Goal
1	Games can be reduced to manipulations of coordinates	Brainstorm own game and create sample coordinate lists for different screenshots in that game.
2	Domain and range; number, string, and image datatypes	Manipulate numbers, strings, and images from their game
3	Define constants, functions, and test cases	Define images for background and all game elements
4	Contracts, tests, functions (the Design Recipe); multiple representations of functions	Move game elements in a simple game (not their own)
5	More practice with Design Recipe and functions	Move targets and dangers in their game
6	Booleans and functions that test values	Determine whether game elements are on- or off-screen
7	Conditionals	Move player in response to user input (key presses)
8	Pythagorean theorem	Detect when two game elements have collided
9	Prepare for their Launch Party!	Polish games and practice explaining them

2.2 Game Program Structure

The game-dissection worksheet and the sequence of screenshot frames motivate the structure of a game program in Bootstrap. Intuitively, a videogame program has two core tasks:

1. Given the current values of each changing attribute, draw the image for the corresponding frame
2. Given the current values of each changing attribute, compute the corresponding values for the next frame

The second task has a bit of complexity, in that the attribute values for the next frame may depend on user inputs, such as key presses intended to control movement of characters. At a high level, however, these two tasks suffice to describe a game. The Bootstrap curriculum decomposes these two tasks into a set of specific functions that students write on their target, danger, and player components. Our decomposition descends directly from Felleisen et al.'s world-programming framework [3], but is tailored to the abilities of middle-school students.

Concretely, over the course of Bootstrap, students will implement functions to (1) compute the target's new x-coordinate, (2) compute the danger's new x-coordinate, (3) compute the player's new y-coordinate in response to key-presses, (4) compute the player's new y-coordinate in the absence of key-presses, (5) compute the distance between the player and the target or danger, and (6) detect collisions based on the distance between game characters. The Bootstrap framework draws the frames, though students could do this step as well if a course has enough time.

Each function demands different programming concepts: computing the new location of an object with linear motion over time requires simple arithmetic functions; computing new locations based on key presses requires piecewise functions that behave differently depending on which key has been pressed (conditionals); detecting collision requires more complicated arithmetic functions about the distance between two game elements; drawing screenshots involves complex function composition to create, scale, rotate and translate images before placing them on a canvas. The Bootstrap lessons, summarized in Table 1, build up to each of these functions after covering the needed concepts from each of programming and mathematics.

2.3 The Design Recipe

Bootstrap takes a particular approach to teaching students how to define functions. The approach is relevant in this paper as it affects both the connections to mathematics and aspects of the volunteer training component. Consider a simple function whose input is the current x-coordinate of a target and whose output should be a new x-coordinate located 5 pixels to the right. Assume we want to call the function *update-target.* Students would develop the function in three distinct steps:

1. Write a *contract*, specifying the domain and range for the function. In this case, the contract is

   update-target : number → number

2. Write a series of *examples* that illustrate the expected behavior of the function. Examples are written using actual calls to the function in the syntax of the language used in Bootstrap. Examples for *new-target-x* include

   ```
   (EXAMPLE (update-target 0) (+ 0 5))
   (EXAMPLE (update-target 5) (+ 5 5))
   (EXAMPLE (update-target 100) (+ 100 5)
   ```

3. Write the function itself (testing it against the examples)

   ```
   (define (update-target curr-x)
     (+ curr-x 5))
   ```

This sequence of steps is called the *Design Recipe* [4]. It helps Bootstrap students think through a function definition problem step-by-step: if they can't articulate the domain and range or write examples, they usually can't write a correct function either. The recipe, however, is more than just a program-design aid: it is also *a key component linking Bootstrap to algebra*. Our description of Bootstrap as going "beyond programming" arises from this recipe and its algebraic connections.

Learning standards for algebra [2] expect students to be able to work with different representations of functions. Three common representations (domain/range, input-output tables, and symbolic form) have corresponding constructs in programming (type specification or contract, test cases, and function definition, respectively). Bootstrap helps students work with all three of these representations, using the concrete context of programming to motivate when and how each representation can be helpful in

problem solving. The fourth common representation, graphs, could also be supported in Bootstrap with more instructional time.

3. THE VOLUNTEER COMPONENT

From the outset, we designed Bootstrap to be taught by technically-qualified volunteers with an interest in programming and community service. Volunteers not only allow us to teach the program in regions far afield from where the Bootstrap design team works, but also provide students with exposure to working computing professionals in various IT-related jobs and careers.

Over the years, Bootstrap volunteers have come from many universities and companies, including IBM, Google, Facebook, Rockstar, LinkedIn, Cisco, NVidia and Apple. Once we have agreements in place to offer Bootstrap with an after-school provider, we recruit volunteer teachers through talks and word-of-mouth at universities and companies in the surrounding area. Each class is staffed by 2-3 volunteers, ideally one with prior experience with the curriculum.

The typical Bootstrap volunteer brings considerable passion for programming, but lacks experience working with middle-school students. Many volunteers initially lack exposure to the structure of game programs and the design practices that underlie Bootstrap (the exception are students from universities that use a similar computing curriculum). As a result, Bootstrap includes non-trivial volunteer training and support component.

The high-level features of our training and support program are:

- A full-day (or two half-day) training class, led by one of the Bootstrap staff, in the city where the volunteers will teach. This class reviews the research behind Bootstrap and its connection to algebra, demonstrates teaching roughly half the curriculum, explains the role of the various worksheets and curricular elements, and provides some basic training in how to manage and motivate students in our target age range.
- Lecture notes that intersperse pedagogic content with classroom management. For example, the following two lecture note instructions govern having students fill in the game-dissection worksheet:

  > [TO STUDENTS] In your groups, take one minute to come up with a complete list of all things in the game. Your group will get a point for each thing they can find. Everyone in your group should have this list written down - not just one person! If even one person in your group hasn't written it down, the group doesn't get the point! GO!
  >
  > [PEDAGOGIC NOTE] *During the minute, walk around and see how groups are doing. This is the time to encourage expectations of community and respect - call out good teamwork when you see it! When time is up, give them a countdown: "30...10... 5... 4... 3... 2... 1... PENCILS DOWN, EYES UP HERE!" Wait for total silence and complete attention.*

- Weekly conference calls between staff and volunteers teaching in the same geographic region; at least one volunteer from each site participates. The calls review each topic in the week's lesson, discuss what did or did not go well, review classroom management issues, and check in on problems from the previous week.

Perhaps our biggest challenge is that our volunteers are excited to teach programming, but not all have bought into following the design recipe and using the worksheets that support the curriculum (most Bootstrap lessons involve 1-2 worksheets that help stage material for students). Like many people with limited classroom experience, our volunteers often lack appreciation for pedagogic approaches beyond instructor enthusiasm. Providing a crash course in pedagogy is therefore one of the key needs of our training program. We discuss this further in Section 5.3.

4. PROGRAM REACH AND IMPACT

We first offered Bootstrap in 2005. Today, we offer 20-30 classes per year across the USA (with growing adoption by teachers in school classrooms). Most of our offerings are through after-school organizations that target under-represented students in urban areas. Table 2 shows course and enrollment data. Roughly 70% of our students get free or reduced-price lunch; most are minorities; roughly 20% have been female.

We have begun evaluating student learning within Bootstrap. Pre- and post-tests on standard algebra word problems and function-composition problems (from actual state math exams) showed statistically significant improvements after a week-long summer-camp offering taught by the Bootstrap founder. Mean scores rose from 51% to 64% ($p \leq 0.007$) on function-composition problems and from 30% to 57% ($p \leq 0.002$) on word problems, both with one-tailed t-tests. Different teachers are using the tests this year, though in formal school settings (where variables including attendance rates and attention to pedagogy are better controlled). Future papers will present Bootstrap's impact on student learning within formal school settings.

Table 2: Statistics on Bootstrap's after-school offerings. The table does not report in-school adoption, which grew from zero in 2010 to 26 classrooms in 2012. The dashes mark years before we recorded enrollment data. The numbers in brackets indicate how many of the classes (from the second column) reported data on student enrollment and retention.

Year	Num Classes	States	Students Started	Students Finished
2005	1	MA	-	-
2006	4	MA	-	-
2007	11	MA	-	-
2008	16	MA, CA	88[9]	62[9]
2009	15	MA, CA, NY	67[6]	40[5]
2010	22	MA, CA, NY, RI	113[8]	103[8]
2011	22	MA, CA, NY, RI, IL, UT, WY	273[22]	240[22]
2012	27	MA, CA, NY, IL, VA, DC, TX, PA, WV, WA	372[25]	121[11] (spring only)

5. LESSONS LEARNED

Bootstrap has been an experiment in using volunteers to deliver content-rich computing curricula through established after-school organizations for at-risk students. Our Bootstrap experience has yielded several lessons about these goals.

5.1 Software Lessons

Most after-school programs we've worked with run out of existing urban (often inner-city) public school facilities.

Computing labs in these schools have predictable limitations: old computers with limited memory, outdated browsers and other core software, and school policies that forbid installation of new software. These constraints demand *web-based programming environments* that work effectively in older browsers.

Social dynamics demand that suitable programming environments use *cloud-based storage* (which often, but need not, comes with web-based tools). The computer labs we use tend to be cramped, offering little room between adjacent computers. This can amplify behavioral problems; if two students with a tendency to fight with one another are at adjacent computers, one of them has to move (this happens often in our experience). Cloud-based tools enable students' work to move with them, both within the classroom and beyond (some students continue working on their games at home or in school or public computing labs). The common alternative, having students bring flash drives, is not practical as younger students forget or lose them regularly.

These two observations led us to build a web-based programming environment, called WeScheme [11] (www.wescheme.org), for Bootstrap. Switching from a desktop IDE to our web-based one also proved crucial to some of our offerings, as it gave our partner organizations more flexibility in where we could hold our classes.

For our target age group (tweens), *good IDEs support not just programming, but also learning*. Bootstrap uses a worksheet to help students through the steps of the design recipe (Section 2.3). This worksheet became much more palatable to students once we built a widget for it into our IDE, rather than having them follow it on paper. In general, students at this age like using the computer, and expect to use it as much as possible. Integrating learning materials into the IDE encourages students to use them.

5.2 Curricular Lessons

By far, our biggest curricular challenge has been accommodating student absence. It is rare that all students are present in any given week. As the Bootstrap lessons build upon one another, we've had to design in ways for students to catch up on a missed session. In addition, with classes meeting only once a week, students tend to forget details even when they were present. Starting each session with reviews of the previous week naturally targets both problems, but this has not proven sufficient.

Gathering all of the worksheets that accompany the curriculum into a single *workbook* per student helps students stay on track (they can refer back to old examples and notes), as well as catch up (one student can look at a friend's completed worksheets while figuring out what they missed). Lesson plans reinforce prior materials by referring back to prior weeks' worksheets as appropriate. The volunteer teachers bring all of the workbooks back and forth to class each week. This seems to contradict our prior recommendation to integrate worksheets into the IDE. In practice, we integrate worksheets that get used often into the IDE and have students use them in the workbook (on paper) once or twice before showing them the computerized version.

The Bootstrap staff also prepared alternate paths to product goals for students who need to get caught up quickly. For example, if a student misses the class on conditionals (needed to get programs to respond to user input), the volunteers can provide a code fragment with the conditional structure, leaving the student to write the test and answer expressions (which they know how to do). This costs the learning goal on conditionals, but helps students remain engaged through a (sometimes partial) product goal. The Bootstrap staff supply these routes to volunteer teachers as needed during the weekly conference calls.

5.3 Managing Volunteers

Over time, we have refined what we cover in the "Teaching 101" component of volunteer training. We also extend teaching lessons into the weekly conference calls. During the calls, we have found that *volunteers struggle to separate engagement from learning* when reflecting on their sessions. Many assume engaged students are learning; some assume that quiet students are not interested. In the weekly conference calls, we ask volunteers to discuss and rate their progress on these goals separately.

Many volunteers underestimate the importance of pedagogic components to the curriculum. They may skip worksheets, or pedagogic instructions in the lecture notes, for the usual reasons that they did not need such structure themselves. We find that most volunteers are much more receptive to pedagogic training after teaching their first Bootstrap class. We therefore *expect second-time volunteers to also attend training sessions*. We also devote more training and support time to explaining how each worksheet and device ties into our learning and product goals.

Sustaining a program through volunteers demands recruiting and retaining them (or at least retaining their interest so that they help recruit others). While recruiting volunteer teachers has not always been easy, it has been easier than expected. We find many volunteers who are *excited to teach Bootstrap because it is more challenging for students* than Scratch or HTML-authoring. In turn, they recruit friends and colleagues. Typically, our pool of willing volunteers exceeds our supply of after-school providers to host the classes. We have certainly lost volunteers who find the learning goals in Bootstrap too rigid for their tastes. However, many of our volunteers are (or were referred by) our own former students who took our college-level courses that used similar foundations to Bootstrap. These volunteers have *an emotional connection to the learning goals, beyond programming and the product goals*. This seems critical to sustaining their engagement.

5.4 Choosing After-School Partners

The Bootstrap staff have explored partnerships with many after-school providers. By-and-large, these organizations have high aspirations for impact on students. These organizations also face considerable challenges, however: they are poorly funded, work through schools with limited facilities, and support students with significant distractions beyond school. Most struggle to find good content for their programs; many are satisfied if they can provide a safe and interesting place for students to spend the afternoon.

Bootstrap's culture differs from typical after-school offerings in two inter-related ways: first, we attempt to go beyond playing with computers to conveying non-trivial content; second, that content, like all STEM content, builds incrementally on earlier content. A class with staged content is challenging to teach in an environment where most students may miss many class sessions.

Based on our discussions with roughly a dozen organizations and teaching partnerships with five, we have identified *key metrics for whether an after-school organization can effectively host* Bootstrap. These include a minimum of 20 hours instructional time, with sessions at least 45 minutes long, at least once a week; attendance rates of 70% or better; and organization staff able to step in and help with chronic or severe behavioral problems among students. While the specific numbers in these metrics might differ in classes other than Bootstrap, we expect similar metrics would govern other programs. While these metrics may seem generous to one who has not worked with after-school programs before, in practice many providers cannot meet them,

and indeed we have had to terminate our partnerships with some (otherwise attractive) providers as a result.

Gathering demographic data and conducting program evaluation can be challenging. In some cases, organizations never provide promised data; several don't gather sufficient data for their own purposes. We often ask our volunteers to gather our enrollment and gender data as a result. The bracketed data in Table 2 illustrate the extent of this problem. Some organizations highly structure student time to achieve solid academic impact on students. This is good, but makes it hard to get additional time for assessment activities (such as pre/post tests, interviews, or focus groups). Expectations on data and assessment are part of our initial negotiations with all potential partners.

6. RELATED WORK

Papert's innovative and inspirational Logo project [8] was an early effort to seriously embrace programming for children. Logo's turtle graphics helps children who don't yet know coordinates perform geometric computations; in essence, children embody the turtle, writing imperative commands that physically perform geometric drawings and computations. Bootstrap differs in two key ways, one tied to each of geometry and algebra. Regarding geometry, we target children who are old enough to work with coordinates; indeed, we seek to reinforce students' existing work with coordinates from math class. Regarding algebra, our reactive and functional programming model allows students to express animations directly as algebraic functions. Logo's imperative computation model does not reinforce the fundamental algebraic idea that functions consume and produce values. From this perspective, it is thus less useful for our goal of helping students understand and master algebra.

Bootstrap's approach to videogame programming differs from those built around Scratch, Alice, and other common early-programming tools. In most languages, game programs consist of small scripts that control each element (in our case, the player, target, and danger). Scripts use imperative programming constructs such as assignment statements and loops to update attributes of elements during gameplay. Bootstrap hides these constructs by having students write programs that generate the attributes for each frame from those of the previous frame. This decision vastly simplifies the programming model—students only need to learn how to define functions and constants and how to use conditionals—but also connects game programming to mathematics standards for algebra and coordinate geometry.

Bootstrap also differs from other common novice programming approaches in our emphasis on up-front design. Scratch and Alice are designed to encourage students to play around and decide what to build they go along (and learn more features). Bootstrap, in contrast, pre-determines the game structure and has students design games within that structure. This lets us integrate our learning and product goals. Our programming model means that students do not write their first animation in minutes (unlike in Scratch). We nonetheless engage students in *creative design* from the outset: the game design worksheet from the first lesson has students envision their games, then gradually teaches them how to build them through programming.

Bootstrap currently lacks the drag-and-drop syntax of most other tools for novice programmers. This is partly by design: textual syntax is closer to what students must master in mathematics classes. A detailed discussion of this issue, while interesting, is out of scope for this paper. The interesting question for this paper is how our use of textual syntax affects the effectiveness of Bootstrap in after-school programs. We are not aware of studies that compare syntax in contexts similar to ours. However, nearly all of our students do complete a working game that they can explain to others after a 9-week Bootstrap course. While syntax might matter, it does not appear critical in the context of a good programming model, compelling materials, and a course structure that requires few syntactic constructs.

The team behind Scratch intersects the founders of the Computing Clubhouse, a network of after-school programs for 8-18 year old inner-city students [9]. Maloney *et al*. studied learning evidenced in student projects with Scratch in the Clubhouse [6], but we are not aware of reports on the interplay between Clubhouse structure and student learning. While many after-school computing programs exist, we have not found any literature reporting on how to effectively design volunteer-led curricula for such programs.

7. CONCLUSIONS

We started Bootstrap with the goal of using volunteers to teach challenging computing classes to at-risk students through after-school organizations. Many USA-wide organizations share our aspirations for quality after-school STEM education [1, 7, 10]. Bootstrap has been successful on several fronts: time and again, novice programmers in our target audience develop their own working videogames within the 9 week class. They find the class engaging, even though our graphics are neither flashy nor sophisticated. They are able to develop programs successfully, even though we use a textual programming language rather than drag-and-drop. We have extensive anecdotal evidence that students gain confidence in mathematics through Bootstrap, both from students and sometimes from their parents.

As our paper explains, however, running successful after-school programs is challenging, especially when they hope to have strong impact on curricular learning (rather than merely offer diversion). In particular, the logistics of after-school programs, and the difficulties faced by after-school organizations, all conspire to increase the challenge of successful delivery. Every aspect, including recruitment, training, attendance, behavior, equipment, and volunteer retention, poses challenges that are often unique to this context.

Nevertheless, we believe Bootstrap offers a positive message. After-school programs can deliver challenging content through which students gain more than just exposure to computing. Many software professionals are eager to work with students, especially if the material feels substantial or reflects their experience. With well-crafted materials, students can perform—sometimes to the joy and amazement of our professional volunteers—activities more commonly associated with real-world software practice, such as extensive testing and live code reviews. After-school offers more opportunities than many computing programs seem to target; with careful management, computing educators could provide significant benefits through this medium.

ACKNOWLEDGEMENTS

Matthias Felleisen has provided insight and inspiration throughout this project. Matthias, Larry Finkelstein, Viera Proulx, and others at Northeastern have provided critical support. Vicki Crosson and Emma Youndtsmith have deftly coordinated the volunteer program. Danny Yoo built the software and kept it running. Dozens of volunteers have donated time, energy, and enthusiasm; we thank them and their employers for making the program possible. We appreciate Greg Morrisett and Stephanie Weirich's support for Bootstrap. We are deeply grateful for funds from Google, Microsoft, the US NSF, Jane Street Capital, the Entertainment Software Foundation, and several private donors.

REFERENCES

[1] After School Science. See http://afterschoolscience.org/

[2] Common Core Standards Initiative. 2010. *Common Core Standards for Mathematics*. Retrieved from http://www.corestandards.org/assets/CCSSI_Math%20Standards.pdf

[3] Felleisen, M., Findler, R.B., Flatt, M., and Krishnamurthi, S. 2009. *A Functional I/O System*. ACM SIGPLAN International Conference on Functional Programming.

[4] Felleisen, M., Findler, R.B., Flatt, M., and Krishnamurthi, S. 2001. *How to Design Programs*. MIT Press.

[5] Jackson, A., Davis, G., Abeel, M., and Bordonaro, A. 2000. *Turning Points 2000: Educating Adolescents in the 21st Century*. A Report of Carnegie Corporation of New York. Teachers College Press.

[6] Maloney, J., Peppler, K.A., Kafai, Y.B., Resnick, M. and Rusk, N. 2008. Media Designs with Scratch: What Urban Youth Can Learn about Programming in a Computer Clubhouse. In *Proceedings of the 8th International conference for the Learning Sciences* (Utrecht, The Netherlands, June 24-28, 2008).

[7] National Parternships for After School Science. See http://npass2.edc.org/

[8] Papert, S.A. 2001. *Mindstorms: Children, Computers, And Powerful Ideas*, 2nd edition. Basic Books.

[9] Resnick, M., Rusk, N., and Cooke, S. The Computer Clubhouse: Technological Fluency in the Inner City, published in: *High Technology and Low-Income Communities* edited by D. Schon, B. Sanyal, and W. Mitchell, MIT Press.

[10] The SEDL National Center for Quality Afterschool. See http://www.sedl.org/afterschool/

[11] Yoo, D., Schanzer, E., Krishnamurthi, S., and Fisler, K. 2011 WeScheme: The Browser is Your Programming Environment. In *Proceedings of the Conference on Innovation and Technology in Computer Science Education* (ITiCSE).