DIGITAL ACCESS TO SCHOLARSHIP AT HARVARD
DASH.HARVARD.EDU

HARVARD LIBRARY
Office for Scholarly Communication

Algebraic Functions, Computer Programming, and the Challenge of Transfer

Citation
Schanzer, Emmanuel Tanenbaum. 2015. Algebraic Functions, Computer Programming, and the Challenge of Transfer. Doctoral dissertation, Harvard Graduate School of Education.

Permanent link
http://nrs.harvard.edu/urn-3:HUL.InstRepos:16461037

Terms of Use
This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. Submit a story .

Accessibility

Algebraic Functions, Computer Programming, and the Challenge of Transfer

Emmanuel Tanenbaum Schanzer

Jon Star
Karen Brennan
Kathi Fisler

A Thesis Presented to the Faculty
of the Graduate School of Education of Harvard University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Education

2015

© 2015
Emmanuel Schanzer
All Rights Reserved

For Lakshyan and Shelley. For Toggle and Fritz. For Nori.

Acknowledgements

This work would not have been possible without the tireless support of Shriram Krishnamurthi, Kathi Fisler, and Matthias Felleisen. Thank you for believing in me all these years, and for always making sure I was thinking three steps ahead. Danny Yoo, thank you for pouring so much of your life into WeScheme. Special thanks go to Emma Youndtsmith and Rosanna Sobota, who took a chance after college to join me on this journey. It is a privilege to work with all of you.

Many have contributed to the work that went into this thesis, and I want to acknowledge the efforts of the teachers and students in this study, as well as the hundreds who I have learned from over the years. I especially want to thank Mrs. Bidwell, Wrenn Goodrum, Mrs. Maybray, Herbert Woodel and Gregory Morrisett – the great teachers in my life, who taught me to see the world through the exacting, playful, and piercing eyes of an educator.

I am grateful to David Perkins, Chris Dede, and Kurt Fischer each served on various committees along the way. You have all shaped so much of my thinking in this dissertation, and pointed me in the right direction. Above all else, I want to thank Jon Star. Thank you for keeping my feet on the ground, for poking holes in my thoughts until I truly understood them, and for your infinite patience with me.

Table of Contents

Abstract

Students' struggles with algebra are well documented. Prior to the introduction of functions, mathematics is typically focused on applying a set of arithmetic operations to compute an answer. The introduction of functions, however, marks the point at which mathematics begins to focus on building up abstractions as a way to solve complex problems. A common refrain about word problems is that "the equations are easy to solve - the hard part is setting them up!" A student of algebra is asked to identify functional relationships in the world around them - to set up the equations that describe a system- and to reason about these relationships. Functions, in essence, mark the shift from *computing answers* to *solving problems*.

Researchers have called for this shift to accompany a change in pedagogy, and have looked to computer programming and game design as a means to combine mathematical rigor with creative inquiry. Many studies have explored the impact of teaching students to program, with the goal of having them transfer what they have learned back into traditional mathematics. While some of these studies have shown positive outcomes for concepts like geometry and fractions, transfer between programming and algebra has remained elusive. The literature identifies a number of conditions that must be met to facilitate transfer, including careful attention to content, software, and pedagogy.

This dissertation is a feasibility study of Bootstrap, a curricular intervention based on best practices from the transfer and math-education literature. Bootstrap teaches students to build a video game by applying algebraic concepts and a problem solving technique in the programming domain, with the goal of transferring what they learn back into traditional algebra tasks. The study employed a mixed-methods analysis of six Bootstrap classes taught by math and computer science teachers, pairing pre- and post-tests with classroom observations and teacher interviews. Despite the use of a CS-derived problem solving technique, a programming language and a series of programming challenges, students were able to transfer what they learned into traditional algebra tasks and math teachers were found to be more successful at facilitating this transfer than their CS counterparts.

Chapter 1

Introduction

Many disciplines rely on *functions* as a foundational concept for thinking about abstraction: a physicist uses functions to describe the movement of a projectile, a chemist to model reactions, and a biologist to reason about population growth. As a gateway concept for so many disciplines, it is no surprise that the course where students are first introduced to functions – algebra – is a key component of standardized tests across the United States. Algebra is a graduation requirement for high schools in many school districts (Loveless, 2008), and studies have found that student performance in algebra is the greatest predictor of future income out of all high school classes (Rose & Betts, 2004). For years, researchers have pointed to functions as an essential concept in math education (Breslich, 1928; Schaaf, 1930), and modern reforms have called functions a "unifying idea in mathematics" (National Council of Teachers of Mathematics, 1989, p. 154).

If algebra is thought of as a gateway class, it is a gate students often struggle to pass through. Our collective fear of algebra is well known, with the phrase "a train leaves Chicago, traveling east at 60mph…" teasing uneasy laughter from most Americans. A recent study (Fong, Jaquet, & Finkelstein, 2014) of California students found that 44% repeated Algebra 1, and that percentage is even higher for traditionally underserved groups. With Algebra 1 being a graduation

requirement, the students who finish high school are by definition more likely to be higher performing math students than their peers. Even among this group, however, a 2012 National Science Board report found that only 76% of graduating seniors even *completed* Algebra 2, despite graduation requirements in 32 states that mandate a third year of mathematics.

Many people see math and computer programming as closely related, and researchers have long viewed programming as a promising domain in which to learn mathematical concepts (Feurzeig, 1969; Khan, 1996; Papert, 1972; Resnick, 2009). The approach taken by many of these studies relies on the achievement of *transfer*: a student's understanding of a particular concept in one domain should translate to better understanding of the same concept in another. Perhaps the best-known example of this approach is found in "Turtle Geometry" (Abelson & DiSessa, 1986), with students directing an on-screen "turtle" to draw shapes using the Logo programming language. If transfer is successful, students should gain a better understanding of geometric properties by experimenting with the routines they have written and then apply that understanding to solve a conventional geometry problem. While some studies have shown successful transfer of various concepts from programming to mathematics (Milner, 1973), transfer has eluded those who wish to teach algebra (Pea, 1984). After much excitement and activity in the 1980s, a survey of the literature concluded that "It may very well turn out

that programming can play a very positive role in mathematics education, but we have not yet done a very good job of realizing that promise" (Fey, 1984, p.261).

Much has changed in the intervening years. Research on transfer has identified specific factors that influence the likelihood of success, and has highlighted the importance of drawing explicit connections between domains. Researchers have examined the cognitive challenges that are specific to algebraic functions and hypothesized precise conceptual barriers than might be prime targets for transfer-based interventions. Programming languages have evolved as well, offering refined tools for expressing concepts mathematically, and enabling activities and interactions that were impossible when Logo began. Recent calls to bring programming into middle and high schools have given the topic renewed energy, funding and political clout. These changes warrant a second look at the challenge of transfer from programming to algebra. However, the majority of recent transfer studies involving algebra focus on software tools for analyzing and manipulating functions (Confrey, 1991; Ellington, 2002; Hegedus & Kaput, 2003; Schwarz & Dreyfus, 1995), rather than programming as a vehicle for building them.

My dissertation is a feasibility study of a transfer-based intervention called Bootstrap, which aims to teach students about algebra through programming. The study uses established measures of algebra performance, as well as interviews, site visits and fidelity measures to determine the strengths, challenges and outcomes

for Bootstrap students. By examining teaching experience alongside student performance, I hope to provide a list of best practices, challenges and pitfalls specifically tailored to Bootstrap but also applicable to other transfer-based interventions.

Chapter 2

Literature Review

Much has been done to understand the challenges students face when learning algebra. The 1990s saw extensive work done around the concept of *function*, both as an abstract idea and as a collection of representations. Work on transfer has also evolved, as researchers uncovered specific conditions under which transfer from one domain to another is more (or less likely) to occur. In a review of the empirical work done around transfer, I will establish the origins of the field, various types of transfer, and the conditions that are known to facilitate or hinder that transfer. I will then examine the domain of algebra, and address the specific reasons why transfer is especially difficult in this domain. Finally, I will consider prior work done to address these challenges, using technology as a means to facilitate transfer.

Transfer

The phenomenon in which skills are said to transfer from one task to another has been observed for centuries. Stemming from a long-held belief that humans possess a common set of general faculties such as "logic," "language," or "creativity," philosophers dating back to Aristotle have argued that a collection of tasks are necessary to stimulate and develop these faculties (Boring, 1950). Those who share this belief would expect that students who learn chess would develop a better understanding of military strategy, or that teaching them Latin would help

them learn another language (Figure 1). The association between task and concept was believed to be direct, and it was thought that a well-rounded education would come from studying a handful of tasks that belonged to each general faculty. In what amounts to a "four food groups" theory of education, it was believed that a smattering of logical, linguistic or creative tasks would appropriately nourish all the basic faculties of the mind.

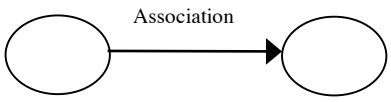

Figure 1 - Direct association between closely related tasks

This way of thinking continued into the early 20th Century. Like Aristotle, Thorndike viewed the mind as a collection of associations, with transfer possible *within* these associations but not between them (Thorndike and Woodworth, 1901). However, he found the scope of association to be far narrower than what the Greeks believed. There was no correlation, for example, between word recall and number recall (Singley and Anderson, 1989). "The mind is so specialized," he concluded, "…that we alter human nature only in small spots, and any special school training has a much narrower influence upon the mind as a whole than has commonly been supported" (Thorndike, 1906, p.246). Thorndike viewed transfer as a two-level problem, with rote tasks occupying one level and the discrete association ranges in the mind occupying the other. For him, and other "associationists" like him, transfer is defined as the presence (or lack) of connection between related stimuli or specific tasks. Absent from the theory is any mention of how knowledge is cognitively constructed, with Thorndike's

association ranges acting merely as *conduits* between tasks, rather than repositories of general knowledge. Without considering how knowledge might be stored or applied, the role of representation is conspicuously absent.

Mental Models

Thorndike's critics pointed out the importance of instruction, citing examples of successful training as a mediating factor for transfer between tasks (Dorsey and Hopkins, 1930; Judd, 1908). In one well-known example (Judd, 1908), two groups of students were taught to throw darts at a target located one foot under water, one of which was given instruction about the phenomenon of refraction. When the target was raised eight inches, the group that received the instruction outperformed the other. This phenomenon (and others like it) suggests that another factor must be at work, which has to do with a particular *mental model* on the part of the learner (Figure 2).

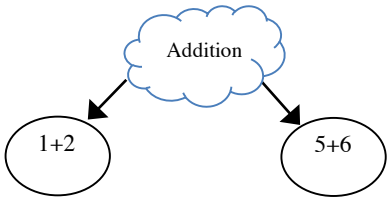

Figure 2 – Two concrete tasks associated with a mental model

If transfer between two tasks is more likely to occur when the learner develops a model that applies to both tasks, it becomes necessary to consider the role of instruction in building that model. In 1945, Wertheimer criticized Thorndike's theory, pointing out that musical melodies are identified not by the individual notes, but by functional relations between groups of notes (cited in Singley, 1989, p.9). This grouping, he argues, is a form of musical representation at work. Wertheimer also experimented with mathematical contexts for the

importance of representation, using a parallelogram-area task that required students to develop representations of parallelograms as simple rectangles that had been transformed in a predictable way. After identifying those transformations, students could provide the area of the parallelogram by instead calculating the area of a simple rectangle. Students were then able to apply this "transformed" representation of shapes to a variety of area-finding tasks, even those that did not use parallelograms.

Having established the role of models as central to transfer, researchers began to study the role of instruction in building those models. At its most basic level, the desired outcome is called Analogical Transfer. We hope that Latin students will see the need for conjugation in French as analogous to Latin, or that the dart-throwers from Judd's (1908) experiments will be just as able to adapt to throwing stones. One of the main factors in analogical reasoning is the *flexibility* of a model, allowing it to be applied in appropriate situations (Reed, Ernst and Banerji, 1974). Judd's students developed a mental model for refraction, Wertheimer's subjects developed a mental model for shape transformations, and both groups were able to apply their models to a variety of related tasks. Transfer, then, is more than a matter of noticing that Task A is similar to Task B. Given the role of instruction as a mediating factor for performance, there must be an intermediate step in which a learner first develops a mental model for "A-like" Tasks, and is then able to apply the model to "B-like" Tasks. As the model

becomes more abstract and flexible, we might expect performance on more and more novel tasks to improve. But what makes for a good model?

David Perkins (2009) explored the notion of model flexibility, distinguishing between *Near Transfer* and *Far Transfer* (p. 112-113). Near Transfer takes place when a model is recognized as being applicable to a very similar task (throwing darts, for example), while Far Transfer involves forming connections between very different contexts, or even different domains altogether. Perkins (2009) cites an example of a classroom where students study the effect of gravity on objects dropped from towers of various heights. The professor hopes that his students will master the underlying concepts of force and acceleration, but the students fail to distinguish these concepts from supposedly unimportant qualities of the sample problems. When the students fail a quiz involving objects falling down holes, one student complains: "all semester, we didn't have any hole problems" (pg. 111). In this example, the students' model for falling objects was too rigid to be adapted to the new task, which Perkins categorizes as a failure to achieve Near Transfer.

These examples illustrate a form of model building that gives us clues into the way students learn. Perkins' physics class falsely integrated the "tower" component into their mental model, which made it inflexible and prohibited transfer to similar tasks (Figure 3). This

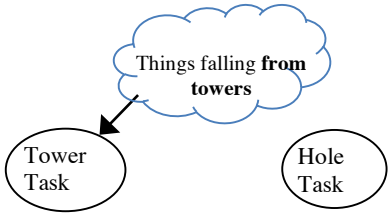

Figure 3 – A mental model for Falling Objects, incorporating irrelevant information (in bold) that prevents application to a similar task.

phenomenon is widely documented in literature that deals with novice vs. expert learning. In situations where the deeper meaning of a representation is obscured, novices will often fixate on less-relevant, surface-level components of a model (Chi, Feltovich and Glaser, 1981; Silver 1979). This phenomenon can hinder performance not only by preventing transfer, but also by facilitating transfer when it is inappropriate. A student may wrongly attribute a rule or idea to some small detail of a particular domain, and then apply that rule to other problems that share the same detail. This phenomenon is described as *negative transfer* (Perkins, 1990). In a study reminiscent of Wertheimer's transfer studies with parallelograms (see above), Schwartz and Yerushalmy (1985) found that geometry teachers typically draw right triangles with one side parallel to the horizon. During assessment, their students failed to correctly identify right triangles that had been rotated, having falsely associated the orientation with the concept and created an inflexible mental model for right triangles. A flexible model is an important condition for transfer, and there is a strong connection between model flexibility and domain expertise.

Unfortunately, the existence of such models is purely theoretical – there is no microscope that allows us to "see" a mental model for dart throwing, refraction, or algebraic functions. What *is* observable, however, is how a learner performs on various tasks. In the concrete realm, we deal with performance over understanding, skills over concepts. A number of alternate theories of learning

exist, which seek to explain the same phenomena while relying on purely empirical data. With their potential to obviate the discussion of transfer altogether, it is worthwhile to consider their position in this review before taking the transfer literature as gospel. So for the moment, let us put aside the notion of mental models and instead begin in the realm of empiricism, shifting our discussion to the intersection of learning and task-accomplishment.

Skill Theories

Rather than guess as to what a learner might *understand*, skill acquisition theories focus on what the learner can *do*. One might imagine a robot programming itself by memorizing rules, associating them based on problem type and forming higher-level rules for breaking down more complex problems. This robot's performance can be measured, and its success or failure on a battery of tasks would allow us to reason about possible gaps in its rule system, without making any statement about what the robot understands.

Gagné (1962, 1968) first theorized the notion of skill hierarchies, in which the acquisition of a set of skills depended on the successful mastery of subordinate sets. Solving an equation, for example, requires a set of smaller skills that include Combining like Terms, Factoring, etc. (Figure 4). Each of those skills can in turn be broken down into sets of subordinate skills, such as factoring constants v. factoring

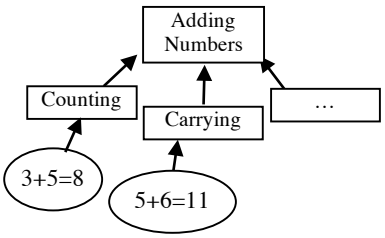

Figure 4 – A higher-level skill, broken up into subordinate skills (associated, concrete tasks shown as examples)

binomials. Gagné suggested that once each skill in a subordinate set was mastered, the larger skill was likely to be mastered as well. In this way, students would build skills that are more general by gradually building up smaller ones. He called transfer within sets *Lateral Transfer*, and the merging of skills into a larger skill *Vertical Transfer*, but note that his use of the term 'Transfer' is much closer in meaning to the Associationists' use of the term: Gagné does *not* employ mental models to explain the phenomenon. Several educational psychologists and cognitive scientists (Bruner, 1996; Fischer, 1980) have supported the notion of skill building through careful identification of related and subordinate skills. Gagné's skill-based definition of Lateral Transfer can be used to explain the dart-throwing example (Judd, 1908) without resorting to mental models: students developed a skill at compensating for refraction at, and were able to apply it to a different depth. This differs from Vertical Transfer, in which Gagné would expect to see the children improve at a wider array of refraction tasks (perhaps one in which the target image was inverted). Skill theory would suggest that sufficient exposure to inversion tasks would increase the likelihood of students acquiring a generalized skill for dealing with refraction, without making any claims about the formation of a mental model for the phenomenon.

In *Mind Bugs* (1990), Kurt VanLehn explores the origins of incorrect and unexpected behavior in the context of a rule-building system. The use of the term "bug" is borrowed from computer programming, which draws a distinction

between random mistakes and encoded, reproducible errors. Indeed, VanLehn finds empirical evidence to support his theory, noting that accidental "slips" are not enough to account for the number or type of mistakes students make. Dynamic Skill Theory (Fischer, 1980; Fischer & Granott, 1995; Yan & Fischer, 2002), developed by Kurt Fischer, seeks to bridge theories of skill development to existing literature on cognitive development, genetics and neuroscience. Fischer broadens the types of association that are possible between skills, incorporating theories from other fields to explain the formation of other connections as a learner matures.

However, there are limits to what can be explained through a skill hierarchy. When a child learns to tie her shoes, she first repeats the steps consciously, one at a time. As her skill improves, the process can be executed faster or with greater flexibility, but there is a side-effect not purely explained by skill hierarchies. The amount of *conscious attention* she needs to tie her shoe decreases, until the process becomes nearly automatic. A strict, observation-only reading of Skill Theory would explain how this skill might feed into a greater understanding of knots, or how its acquisition would eventually lead to greater flexibility with lacing other footwear, but would fail to explain the reduction in cognitive load. Related phenomena can be found in memorization tasks, in which subjects are able to group items in such a way as to maximize recall (Gagné & Glaser, 1987. p. 54). Without giving up their position as strict empiricists, skill

theorists refrain from postulating about the mechanism by which this "chunking" phenomenon occurs (Figure 5). However, even acknowledging the phenomenon suggests the operation of an internal, unobserved mechanism. While the locus of Skill Theory remains firmly rooted in the development of skills as they pertain to task completion, it is clear that there are compelling reasons to shift this focus inward, and to consider where and how these skills might be stored in the mind.

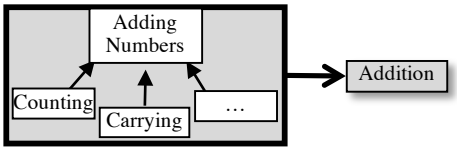

Figure 5 – A skill hierarchy becomes chunked into an atomic skill

Cognitive Architectures

Theories of cognitive architecture seek to map the observed process of skill acquisition to a brain-based theory for program storage, retrieval, and execution (Anderson, 1976; Newell & Simon, 1972). Because these theories can be modeled as computer programs that "learn," many of the theories in the field have produced software whose behavior can be compared to human learners. Examples include the collection of ACT-derived systems (Anderson, 1976, 1983; 1992; Anderson, Corbett et al, 1995) and the Soar architecture (Rosenbloom, Laird, et al, 1985), both of which have been employed as cognitive tutors and learning tools across a variety of domains. As with model-based theories of transfer, Cognitive Architectures assume that some intermediate representation of knowledge exists in the mind of the learner, and that the representation grows and evolves through

experience. By making these models testable and concrete, these theories manage to avoid some of the criticisms of purely theoretical, model-building frameworks.

A child who is learning to tie her shoes may begin by memorizing a list of instance-specific steps, storing them away and then repeating them the next time she is asked to tie her shoes. She may be able to recall this list whenever she is asked to tie her shoes, but be completely flummoxed when presented with two pieces of rope that must be joined together. According to the cognitive architecture literature, this child has stored the steps and context for shoe-typing in what is called *declarative memory* (figure 6). This memory may include information that is irrelevant to the task, as we saw in the Tower v. Hole confusion posed by students in Perkins' example (2009). She may, for example, assume that the first step is to put her shoes on, and to sit comfortably on the floor. Declarative knowledge is brittle, and can only be applied only if the new task is identical to one that has been seen before. *Procedural memory*, on the other hand, parameterizes these steps as being associated with a goal (e.g., "knot tying"). Anderson (1983) describes declarative memory as stored facts or information (the "what") and procedural memory as goal-oriented statements (the "how"). To solve a problem, a learner may call upon declarative knowledge of similar problems or facts, or procedural knowledge of strategies. Procedures are characterized by their flexibility and speed, reminiscent

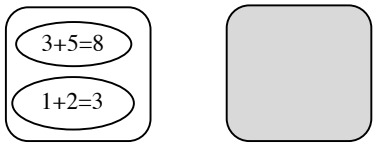

Figure 6 – Two summation-by-counting tasks are stored in declarative memory (procedural memory – on the right – is shown as empty).

of “skills” or “models” discussed above. Procedures can be adapted to a variety of tasks, and procedures can be combined to form higher-level procedures. A learner may execute a procedure without ever being consciously aware of the steps he or she takes, which explains the chunking phenomenon described earlier. However, where Skill Theory merely acknowledges this experience, the Cognitive Architects go to great lengths to analyze the process.

Anderson et al. (1995), describe the process of shifting knowledge from the declarative to procedural store as “knowledge compilation” (p. 169), in which the learner generalizes over examples stored in declarative memory, creating a procedure to achieve a task (Figure 7). The exact steps of the procedure are also stored in declarative memory, but the *goal* is stored in procedural memory. When confronted with a similar task in the future, the learner need only match the goal at hand with goals stored in memory, then access and execute the appropriate procedure. Procedures can also be compiled together, to create a more general procedure with a more widely applicable goal. For our purposes, knowledge compilation serves as the mechanism by which skills are generalized and mental models are developed.

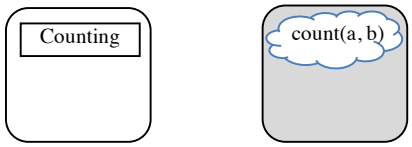

Figure 7 – The tasks are **compiled** into a procedure, which is stored in Declarative Memory, with the “counting” goal stored in Procedural Memory.

In their study of algorithmic problem solving, Newell and Simon (1972) observed a similar trajectory to Anderson's: learners would first blindly repeat a list of steps, then memorize those same steps, and finally internalize them in a more compact representation. However, they also point out that the compilation process results not only in efficiency and flexibility gains, but also in an increased understanding of the algorithm itself (p. 822). It is interesting to note that the term has the same meaning in computer science as it does in cognitive science: compilation is a process through which a verbose structure is boiled down to a simpler one, and along the way something is learned about the *meaning* of the original structure. This compilation occurs constantly, as new subroutines are added, trimmed, or optimized, and the goals become more flexible and more adaptable.

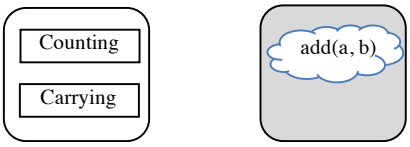

Figure 8 – Carrying is compiled as a procedure too, and the goal stored in procedural memory has been generalized to include both counting and carrying subroutines

Synthesizing Transfer

Model-building theories focus on the way knowledge is represented in the mind, highlighting the importance of a model that accurately represents a concept and can flexibly be applied to relevant tasks. Skill Theories, on the other hand, use task performance to identify the hierarchical connections between related skills, emphasizing an empirical approach that tracks the development of rules through experience. These theories are orthogonal, rather than mutually exclusive. If skill

acquisition is constructed as a form of programming, what is the nature of the computer on which these programs run? In what machine might mental models be stored, modified and retrieved? Drawing from a number of diverse, learning-related fields, the study of Cognitive Architectures offer a possible bridge between these two frameworks.

Cognitive Architectures propose a mechanism by which learners gradually integrate information they have seen into concepts they understand. A learner may begin by memorizing the steps and information involved in a task, allowing them to carry out the task through brittle, rote repetition. Through experience, they compile the list of steps into a procedure, replacing the memorized list with a flexible goal. It is *Compilation* and *Goals* that form the bridge between skills and models. When closely related tasks are stored as a compiled procedure, this phenomenon may be considered "near" or "lateral" transfer. When individual procedures are compiled into a more general one, we find links to Gagné's description of "vertical transfer". As suggested by Skill Theory, this compilation results in accelerated execution, and subordinate procedures that are used often are more likely to be compiled into a more general, higher-level procedure. Mistakes in compilation may occur as well, resulting in negative transfer and reproducible slips, à la Van Lehn's *Mind Bugs*.

Like the model-builders, Cognitive Architects assume that these procedures actually exist somewhere in the mind, and are shaped by experience. For example,

a procedure for falling-body problems is shaped by the experience of watching an objects fall from towers or down holes, from learning about gravity in physics class, etc. When we talk about the fitness of these procedures, we consider the degree to which they accurately describe the structural properties of particular concept. Does a students' "falling-body procedure" include general parameters for distance and acceleration, or has irrelevant information about towers been compiled in by mistake, as a form of Perkins' "negative transfer"? Is the goal flexible enough to be applied to problems that deal with holes?

My use of the term "transfer" is closest to the concept of knowledge compilation. It is both a singular occurrence (the immediate recognition of one task's similarity to another) and a permanent change (the compilation of that new information into a new procedure). When I refer to a student's "model" for functions, I describe the structure of goals and procedurals that is constantly being revised, rebuilt and refactored, *and* the associated goal that signals to the student that this structure should be applied to a given task.

This compilation is influenced by experience: rich models are more likely to be formed by completing rich tasks. But what defines a "rich task"? A rich task may have many valid solutions, but only a handful of optimal ones. The game of chess is easy to learn, but difficult to master (Singley, 1989, p. 29). Singley points out that chess is about picking an optimal move out of many valid ones – and the intuition for such a move is built after thousands of hours against multiple

opponents. Perkins (2009) draws a similar conclusion when he advises learners to "play out of town," emphasizing the importance of applying a concept in diverse settings in order to deeply understand it. Like a Grand Master's understanding of chess, a rich model is built through vast experience in a variety of settings.

However, merely completing a rich task does not guarantee that compilation will result in a rich model. Cognitive Architectures include limits on working memory, and there is evidence that compilation imposes its own memory overhead. Reed, Ernst and Banerji (1974) studied the relationship between memory and compilation by having students complete the closely related "Missionary-Cannibal" and "Jealous Husband" puzzles (p.438), both of which are known to strain working memory. The solutions to both puzzles are similar, and the authors expected their subjects to transfer what they had learned when solving one puzzle to the next. What they found instead was that few subjects recognized the structural similarities between the problems, even after completing them. The first puzzle was sufficiently complicated as to prevent compilation, and even after solving it successfully students were no better off when solving the second one.

Educators must strike a balance between task richness and working memory. Simple tasks are unlikely to result in complex models, but complex, monolithic tasks may be too large to be compiled at all. To consider the challenge of formulating a rich task in algebra, it is necessary to understand the wealth of concepts at play when dealing with functions. In the next section, I will discuss

various representations of algebraic functions, and how their interaction forms a rich model for algebra.

The Algebraic Domain

From an early age, children are presented with mathematical notation like "2+3" or "5+x=10". The implication is that math is a *process*, and that mathematical expressions are essentially questions for which the student must find an answer ("5" and "x=5", respectively). When they enter an algebra class, however, math is constructed quite differently. The notation "f(x)=x+10" will be confusing to a student who is looking to find an answer, because there is *no question*. Instead, the notation defines a mathematical object: the function *f*. Rather than asking for students to calculate an answer, an algebra teacher asks their students to reason about their properties (Breidenbach et al., 1992; Sfard, 2000). A student might be asked to identify the y-intercept of a function, how many roots it has, or whether or not it has an asymptote.

An Object Model for Functions

At the very least, a model of functions requires that students develop an ontological understanding of processes *as* objects. This shift requires a level of abstract thinking beyond what is required for arithmetic (Piaget, Grize, Szeminska et al., 1977). Unfortunately, this switch is decidedly nontrivial. Sfard and Linchevski (1994) explored the challenges of viewing functions as objects and

processes, which they call “Reification”. They define reification as “our mind’s eye’s ability to envision the result of processes as permanent entities in their own right" (p.194), the ability to name a process and to recognize that an arithmetic relationship can be viewed as an entity by itself. This shift requires that students cross a “cognitive gap” (Herscovics & Linchevski, 1994, p. 63), in which named objects (“*f*(x)”) are also thought of as processes acting upon unknowns (“2x+4”). A related construction is found in Slavit’s (1997) “Property-Oriented View” of functions, in which the objects are made concrete by considering various properties that are preserved across transformations. This also mirrors the approach taken by Schwarz and Dreyfus (1995), who focus on the transformations rather than the properties they preserve. Despite these differences in focus, these characterizations are not mutually exclusive (O’Callaghan, 1998). All of them require the formation of an *object model* for abstract arithmetic processes, as opposed to the strict application of those processes to find an answer. For students who still think about “doing math” as “getting the answer”, questions about properties of a function are not just difficult – they are beyond the limits of the students’ conceptual model.

Once this gap has been crossed, the objects must be fleshed out to gain *properties* (e.g., linearity, number of roots, etc.), and students are asked to reason about these properties. This closely parallels the notion of goals being associated with compiled procedures: a goal must be flexibly defined in order for the

procedure to be flexibly applied. Asking a student to identify the roots of a function, for example, assumes that they understand functions as objects *and* that "root-having" is a property of those objects.

Moreover, these properties may be discussed in a variety of representations, with students thinking of functions as lines on a graph, tables of inputs and outputs, symbolic formulas, or a mapping between domain and range. In this construction, it is necessary for a student to understand the representation in order to decipher that which it represents. This definition is closely hewn from that of Kaput (1991), who writes about notational systems that are "materially realizable cultural or linguistic artifacts shared by a cultural or language community" (p. 55). It is because we agree on conventions for graphing, for example, that we are able to discern the meaning of a graph. Slavit, in his argument for a property-oriented view of functions (1997), addresses the problems of notational settings, again referring to the mix of terms and symbols that define a representation.

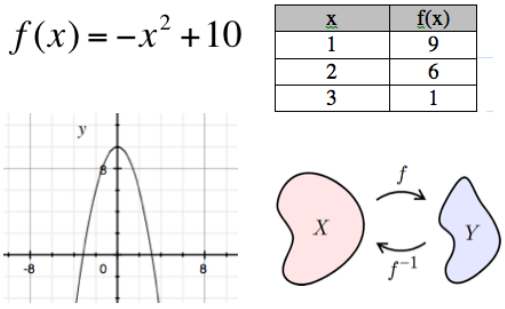

Figure 9 - Four representations of a single function

Algebra textbooks vary somewhat in the number, type and order of representations they use (Dreyfus & Eisenberg, 1982; Fey, 1984), but there exists a core set of representations that I will draw from in this study. This set finds common ground in the literature, but is also endorsed by modern standards

(Common Core Standards, 2010; Mayer et al., 1995 NCTM, 2000): Tables and Function Machines, Graphs, Sets and Symbolic Equations. It should be noted that this corresponds exactly to Kaput's "Big Three" (1998, p. 272), with the addition of the Set-Mapping representation. Each representation has its own properties, vocabulary and operations – all of which must be incorporated into a complete model for algebraic functions.

Tables and Machines

The idea of the "function machine" as an entity is recommended as an ideal way to introduce students to functions (McGowen et al, 2000; NCTM, 2009). The machine itself is the Object, while the inner workings symbolize the Process (Slavit, 1997). Wilson (1994) shares a student's explanation of functions, which mirrors this representation: "What I mean ... is with the f, the f(x), you have to put, it's like a grinder. You put a number in right here and this f is going to change this x in some way" (p. 354). This representation is analogous to lists or tables of ordered pairs (p. 261), by emphasizing the discrete inputs and outputs of a function.

Graphs

In contrast to a table's emphasis on input-output pairs, a graphical representation highlights the behavior of a function over a range inputs. Graphical representations rely on a large number of inert skills, which must be acquired

before the underlying mathematical concepts can be explored (Curcio, 1987, p.383; Mevarech & Kramarsky, 1997). Students must understand what the axes, origin and coordinates represent, and be familiar with the visual concepts of points, scale, and magnitude in order to reason about a graph, plot a point, or locate a root. Once they have mastered the skills, however, students can use this representation as a powerful tool for analyzing the behavior of a function over groups of inputs, identifying properties such as roots, maxima and minima, etc.

Set Mappings

Functions may also be represented a mapping between input and output sets (Slavit, 1997). In this representation, the set of all possible inputs is transformed to the set of all possible outputs by the behavior of the function. When describing this transformation, students must determine if the mapping is "one to one" and "onto" as they consider the properties of the function. The representation of a function as a mapping from one set to another is familiar to many, as the concept of related quantities is rooted in everyday experience (gallons of milk per dollar, miles per gallon, etc.). Ponce (2007) suggests that a 'mapping' representation is relatively intuitive, and forms the "first core idea" in his survey of functional concepts.

Symbolic Equations

When represented as series of symbols, a function can be construed as a sentence waiting to be filled in, with variables being replaced by values in order to provide an answer. These "math sentences" can be composed with one another to generate new sentences, and can be manipulated by many of the rules students have learned from arithmetic. For students who are new to algebra, this representation allows them to focus more on the familiar process, and largely ignore the object notion (Usiskin, 1998).

Rich Tasks for Algebra

The two problems for model building outlined above involve (1) building an object-based model for mathematics that includes all relevant properties, and (2) making that model flexible enough to span multiple representations. A rich model should include multiple properties or representations of a function object. As we have seen from the transfer literature, abstract rules such as light refraction or chess strategy may be easy to memorize and recall, but can be difficult to apply in novel situations. Singley (1989) and Perkins (2009) emphasize the importance of concrete experience in overcoming this obstacle, suggesting that an instructor should use tasks in which students are able to apply their understanding of functions in a concrete way. Teachers often seek out real-world tasks to provide this experience, such as the relationship between gears on a bicycle or the position

of a car as it changes speed. This allows students to refer to a concrete base of knowledge, and use that as a check against functions they have built or manipulated.

Word problems are a complex application of function models, in which an informal description of processes must be formalized or solved for a given constraint. Despite having arithmetic solutions, the challenge of word problems is about identifying the functional relationships embedded in a narrative, and potentially bringing to bear a number of properties and representations to arrive at a solution. Consider this well-known word problem:

A train leaves Chicago for New York, traveling at 75 mph. At the exact same time, another train departs New York for Chicago, traveling at 60 mph. If New York and Chicago are 700 miles apart, how long will it take the trains to pass each other?

The goal of this puzzle is clear, yet there are no instructions for which path will take us there. A student can use simple arithmetic to determine the location of each train every hour, as they seek the time when both trains have the same location; they could formalize the relationship between train-distance and time using a formula, or by drawing a line on a graph. Each approach has its own strengths and weaknesses, and the freedom to select an appropriate representation requires students to marshal their understanding of each representation. As with

chess games or logic puzzles, word problems like this one allow students to choose an optimal solution from several valid ones. The ability to derive the functional relationships at work in unstructured word problems can be seen as a perfect rich task: to solve the problem, a student's goal must be flexible enough to apply the appropriate model, and the model itself must be rich enough to encompass the conceptual depth of the task at hand. However, we know from our synthesis of the transfer literature that this richness comes at a price.

Obstacles to Compilation

In order to be compiled, a rich task must fit within working memory, leaving enough space for compilation overhead. Unfortunately, this is a particularly difficult challenge for algebraic functions. Each representation includes a large collection of specific vocabulary, mechanical skills, and concepts. For example, students must learn how to draw and label their axes, plot points and understand words like "scale," "origin," "slope," and "root" just to work comfortably with graphs. How much "graph knowledge" are they able to employ when solving symbolic equations? Additionally, the amount of raw computation necessary to convert one representation to another makes the task prohibitively time consuming for students, forcing them to focus on the rote computation rather than the higher level task at hand. Schwarz and Dreyfus (1995) found that "beginning students cannot be expected to simultaneously think at the higher level needed to decide which bounds and scales are appropriate to a given task" (p.

263). Given the wealth of knowledge students must engage in rich algebra tasks, it is no surprise that these tasks are difficult to complete, and unlikely to result in compilation.

This forces teachers to focus on tasks that deal with a single representation, resulting in predictable learning outcomes. Students may be given activities that stress their knowledge of tables, graphs, or formulas, but never more than one at a time. While these tasks may be challenging by themselves (Kieran & Sfard, 1999; Kirshner, 1989), we would expect the focus on individual representations to result in the compilation of separate, disconnected models. Indeed, a wealth of studies shows that students struggle to see the connections between representations (Bloch, 2003; Campbell, 1992; Charles & Silver, 1988; Resnick & Ford, 1981; Wagner & Kieran, 1989). Schwarz & Dreyfus (1995) found that students who are highly proficient when working with one representation of functions might be completely unable to reason about another. Likewise, the large number of prerequisite skills involved in graphing may explain why a student's understanding of graphs translates so poorly into other representations (Carreira, 2002; Cunningham, 2005; Dreyfus, 1999). Wilson (1994) found that even teachers themselves are often unable to explain what a graphical property (such as the vertical line test) means outside of the graphical context. For many students and teachers, the symbolic notation is felt to be the "real" function (Carriera and Evan, 2004), which can be frustrating for students who are uncomfortable with this

representation. Because of the bias towards symbolic representations, "persistent algebra errors may reflect disengagement from declarative content rather than inability to deal with it" (Kirshner, 2004).

In each of these situations, students have developed representation-specific models for functions and are unable to compile them into a unified model. Like the physics students struggling with "hole problems," these students have falsely identified the representation as a necessary condition for applying each model. Perkins (2009) calls this process "conceptual welding," in which the various representations become inseparable from the mathematical concepts they represent. This aligns closely with Van Lehn's compilation errors, in which an erroneous facet is compiled into a procedure, preventing it from being flexibly applied. To address these challenges, teachers must find ways to optimize rich algebraic tasks, reducing demands on working memory and making compilation possible.

Optimizing Rich Tasks

Teachers employ a number of strategies to optimize these tasks for clarity. One of the most common is the use of real-world situations for rich tasks, which allow students to link the behavior of a function object to a concrete experience. For example, asking students to consider the trajectory of a baseball can easily describe a quadratic relationship, allowing students to rely on a familiar experience. Another popular technique is demonstration by example: teachers will

often walk through the application of a function on a number of inputs, to demonstrate exactly how the object behaves. Likewise, it would be helpful to explicitly connect representations, allowing students to manipulate a function in one representation and see their changes reflected in another. Teachers may offset the computation load of converting between representations by doing the work themselves, or by providing students with a formula, a table and a graph as part of an assignment. This echoes the recommendations for explicit connections (Bransford, 1999, Bransford, 2005; Singley, 1989; Van Eck, 2001), as is it now frequently suggested that representations should be taught in parallel or explicitly connected by the instructor (Bazzini, 2001; NCTM, 2000; Resnick & Omanson, 1987).

Unfortunately, these techniques only go so far. It is prohibitively labor intensive for a teacher to check the behavior of multiple functions at once, or to quickly translate a single function across multiple representations. Asking students to incorporate these calculations into their own practice might be helpful, but it only increases the number of steps necessary to complete the task and makes compilation less likely. Word problems help *relate* functional relationships to a concrete domain, but students are confined to the abstract when trying to solve them. Finally, there remains the ontological problem of functions as objects: students who calculate an answer using rules and conventions of the algebraic domain may not understand anything about the functions themselves. As with long

division, correctly calculating an answer does not necessarily give students a sense for the underlying concepts at work. Having established the specific challenges faced by teachers and students around rich algebraic tasks, we can now turn our attention to the ways technology has historically been used to address them.

The Promise of Technology

Technology aims to reduce the cognitive load of rich algebraic tasks, improving the chances of compilation. While the scope of algebra-focused technologies is broad, all of these tools offer a mix of four specific advantages, which address the challenges outlined in the previous section:

1. **Rapid Computation** – students are free to spend more of their time on *analysis* rather than calculation, which may reduce cognitive overhead by shrinking the amount of time and effort needed to generate a graph, complete a table, etc.
2. **Rich Visualization** – programs can be used to display functions in multiple representations simultaneously, or to switch back and forth rapidly between them. Comparing these representations side-by-side may reduce memory overhead.
3. **Concrete Examples** – software can be used to demonstrate functional relationships through multimedia, which may preserve real-world connections (e.g., actually seeing two trains move towards one another, modeling a situation described in a word problem).
4. **Conceptual Similarities** – the act of computer programming can employ concepts such as abstraction, variables and functions. If a programming language is sufficiently similar to algebra, rich programming tasks might serve as a viable surrogate for rich algebra tasks.

These propositions have made so-called "cognitive technologies" (Pea, 1987) an attractive fit for educators struggling to balance richness with clarity

(Mayer, 2005; Marzano, 1998). Graphing calculators, for example, have become a staple of the modern-day algebra classroom (Berger, 1998), allowing students to see a function translated across representations without any computational overhead. Programs like Mathematica and Maple offer sophisticated tools for building and exploring functions, with instant computation designed to free up cognitive load and facilitate compilation.

While these tools may reduce the transfer barriers between functional representations, they introduce barriers of their own. Each tool brings its own conventions, vocabulary and skills, which creates familiar the risks of conceptual siloing and welding: can students connect operations in a computer program to the algebraic ones they represent? If they build a rich model by solving word problems with a calculator, can they apply that model when the calculator is taken away? As we consider the various technologies in this section, we must bear in mind the added cost of teaching students to use each tool, as well as the potential for the tool to become a problem solving crutch.

The previous section defined the constraints of a rich model for functions in algebra: a concrete understanding of functions as objects, and the ability to view these objects in multiple representations. All of the technologies designed to address these constraints allow students to both create functions and then view them in multiple representations, but each one makes tradeoffs that it better suited to one challenge or another. Graphing calculators, for example, impose tight

restrictions on how students define functions (e.g., enforcing specific names for variables or functions, rejecting mathematically-valid equations that fail to match a specific format, limiting functions exclusively to the domain of numbers, etc.), but allow them to explore those functions flexibly in multiple representations. Tools like Logo, on the other hand, have far fewer restrictions on how functions are defined, but require a great deal of extra work to view these functions across traditional representations. While each tool attempts to straddle the line in a different way, it is useful for our discussion to group these technologies into two categories, following the dual challenges of *representing* and *reifying* functions: those whose primary focus is the *visualization* of functions, and those whose focus is on *programming* them.

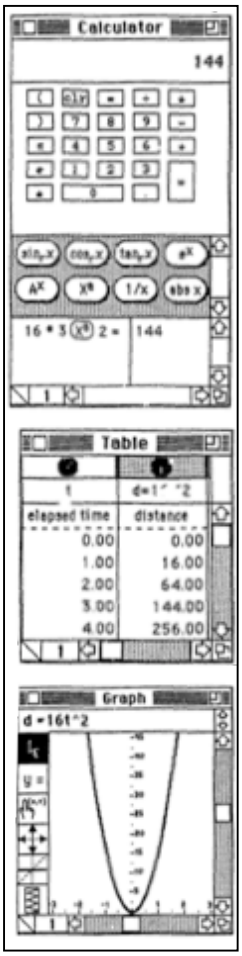

Figure 10 – The Function Probe interface

Visualizing Functions

Perhaps the best-known example of technology as a means of visualizing functions is the graphing calculator. On a graphing calculator, students define a function in one representation (symbolic), and then see the corresponding graphical representation instantaneously. Other visualization tools offer variations on this theme, allowing students to define functions in one way and instantly them using a variety of representations utilizing the rapid computation afforded by computers. In the

CARAPACE software program (Kieran, Boileau and Garançon, 1996), students define functions in a language-like interface (for example, "`price of pen × 3 gives price of watch`" (p. 266)) rather than using pre-defined symbols, and can then explore various representations of those functions. The Triple Representation Model (TRM) tool also focuses on displaying multiple representations, allowing students to see changes translated across each representation in parallel (Baruch & Dreyfus, 1995). Jere Confrey's Function Probe (1991, 1996) adds a unique feature that allows students to define a function by recording sequences of calculator operations, and then store them as a single button on a simulated calculator, essentially chunking a process into a single object. Like the other tools in this category, these objects can then be explored across multiple representations.

SimCalc (Kaput, 1998; Hegedus and Kaput, 2003) adds *concreteness* to the computation and visualization benefits of technology. As with programs like TRM and the Function Probe, SimCalc allows students to view multiple representations of a function at the same time. However, SimCalc is unique in that it also allows students to model behaviors using the functions they have defined, and see these behaviors animated on-screen. A student who models the position of a train over time can

Figure 11 - The SimCalc interface

import an image of the train, and then *watch it move* across the screen in accordance with their function.

This modeling functionality makes SimCalc unique among visualization tools, allowing it to address some of the ontological challenges of function objects. Functions become the blocking instructions for "actors" on a "stage," putting students in the role of a director who decides how their actors should move. This provides not only an engaging visual experience, but also a connection to real life events. The modeling activity forces students to consider behaviors in their own right, and to abstract individual calculations into rules and patterns. SimCalc also provides rich tools for *sampling* these simulations, displaying a graph or table of position of the train's position with respect to time. This puts the concrete nature of physical phenomena first, and shifts the focus from "multiple linked *representations* to multiple linked *descriptions* of real situations" (Kaput, 1998, p. 275).

Research on the learning impacts of these tools is mixed. Graphing calculators, by far the most widely adopted technology, have surprisingly little data to support their use. A meta-analysis of graphing calculator evaluations (Berger, 1998) found many problems in experimental design, with studies lacking a control group or a proper form of assessment. Other studies are more promising: a single-classroom study of the Function Probe (Confrey, 1996) found marked improvement in students' ability to connect functional representations and a more

flexible mental model for functions, while two classroom studies of TRM (Baruch and Dreyfus, 1990 and 1995) also found strong gains on transfer tasks. While both of these studies used control groups, the individual assessments were carefully tailored to the tools themselves, making it difficult to gauge transfer into conventional tasks. Moreover, the small sample sizes and reliance on a single teacher make it difficult to generalize the conclusions made in these studies.

Studies of SimCalc, by contrast, have consistently utilized rigorous experimental design, traditional math assessments and large sample sizes. In a study conducted with grade school students in Texas (Roschelle, Tatar & Shechtman, 2007), SimCalc was used to replace a multi-week unit in a traditional mathematics curriculum for more than 800 students. Those in the treatment group made much larger gains than the control group when dealing with questions that required them to reason about the behavior of a function across multiple representations. These gains were shown to be significant across teachers, strengthening claims about the tool's effectiveness. A multi-year study in Massachusetts high schools (Hegedus, Dalton, Kaput et al., 2007) demonstrated similar gains.

Unsurprisingly, these studies provide evidence of both siloing and welding. Because of their focus on exploring functions rather than defining them, these tools require students to define functions in an inflexible format. A student who wishes to model the location of a moving train, for example, must still derive the

function on their own before entering it into their graphing calculator or software program, and must do so using the conventions or syntax of that program. The risk of siloing lies in the requirement that students translate their understanding of a task into a specific representation *before* entering it into the tool. As a result, it may be difficult for novices to distinguish between the limits of the tool and the limits of the math it is designed to explore. If a calculator is unable to solve a specific expression, is that because the expression is entered incorrectly or because there is no solution? Indeed, several studies have found that the restrictions of the calculator interface constrain the ability of students to directly access the material (Berger, 1998; Drivjers, 2000; Zheng, 1998). The authors of CARAPACE's natural-language interface for defining functions found that students were unable to connect the underlying mathematics to the English-like statements they had defined in the software (Kieran, Boileau and Garançon, p. 267). Meanwhile, a meta-analysis of impact studies found that any positive effect from graphing calculators disappears when students are assessed without them (Ellington, 2003). If students *are* able to build a better model with these tools, they are unable to apply it without them. In their concluding remarks, Kieran, Boileau and Garançon conceded, "not all technology supported roads that are intended to be algebraic reach developing meaning for traditional algebraic representations and transformations." (p. 267).

Intermezzo

The gains demonstrated by users of SimCalc are encouraging, and deserve further discussion. What distinguishes SimCalc from other exploration-focused tools is its use of *modeling*, which forces students to think concretely about functional relationships within real-world phenomena. In this way, SimCalc may be considered a limited programming language, tailored exclusively to animation of objects using a narrow class of functions. SimCalc is the only tool in this category that seeks to make functions concrete, and its emphasis on representations as linked descriptions of program behavior is directly in line with best practices involving transfer and compilation. However, SimCalc's modeling environment imposes a constraint on the *content* being modeled. While phenomena involving position, velocity and acceleration are easy to model, a teacher would struggle to model other relationships, such as those between price and profit, heat and pressure, or radius and circumference.

The limitations each of these tools impose on the definition of functions or the ways they are represented should give us pause. TRM, the Function Probe, and Graphing Calculators force students to translate from real-world phenomena to pure mathematical forms before they can be explored. SimCalc attempts to bridge this divide by allowing the tool to visualize the phenomenon itself, but places deep restrictions on what can be visualized. When viewed as a programming tool, the primary drawbacks of SimCalc involve the limitations of the language for writing

programs and the visualization of their output. These limitations are understandable, as the goal of SimCalc is to explore algebraic concepts from a familiar mathematical domain, while minimizing the overhead necessary to use the tool. But at this point, it is fair to consider whether the domain of algebra *may simply be too complex* to be explored in this way. Analogies are often used to explain difficult concepts, by embedding them in a more accessible or familiar domain. For algebra, the most commonly explored domain is computer programming, which has been used as a source domain in transfer experiments going back nearly half a century.

Programming Functions

At first blush, programming and algebra have a great deal in common. A programmer identifies sequences of instructions and combines them into a function, just as a mathematician combines a sequence of calculations into a function object. Given their similarities, it is no surprise that researchers have long considered programming as a possible "surrogate domain" for algebra. Teaching algebra was one of the original design goals of the Logo programming language (Milner, 1973), and many researchers have studied it as a vehicle for algebra (Clements, 1985; Feurzig, Papert, Bloom et al., 1970; McCoy & Burton, 1988; Milner,

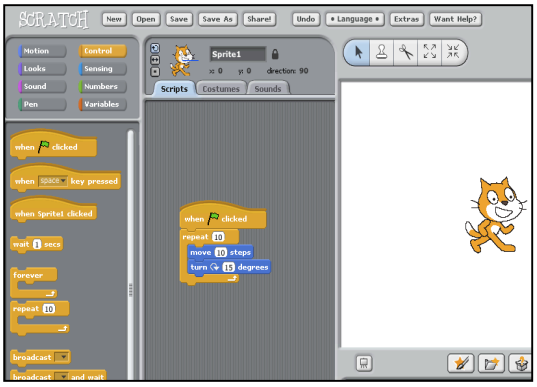

Figure 12 - The Scratch programming environment

1973; Noss, 1987). As Logo waned in popularity, other introductory-programming tools rose to take its place, such as BASIC (Hatfield and Kieran, 1972), Interactive SET Language (ISETL) (Breidenbach, Dubinsky et al., 1992), ToonTalk (Kahn, 1996), Boxer (diSessa, 2000), ALICE (Cooper et al., 2000), Scratch (Resnick, Maloney et al., 2009) and others. Some of these tools were, like Logo, explicitly designed to teach mathematical concepts. Other tools were focused elsewhere, but teachers and parents maintained the expectation that all programming is deeply connected to mathematics: if a student learns to program, the thinking goes, surely they will get better at math in the process.

In contrast to visualization tools, the value proposition of programming is that it shifts the learning of mathematics from the task of *analysis* to that of *experience* (diSessa, 2000). A particular strength of programming is its use as a modeling tool, in which students are able to model complex mathematical relationships to explore concepts such as statistical distributions, the movement of gas molecules, etc. (Wilensky, 1997). As with SimCalc, students engage in the act of creating worlds governed by functional relationships, defining the rules that control a simulation they observe and modify.

Ultimately, the hope is that students will be able to transfer the rich model they compile while building and manipulating these worlds back into the domain of algebra. Papert (1972) famously argued that children would develop a "mathematical way of thinking" that would then transfer into their conventional

math classroom, simply by learning to program in the language. Noss (1986) explains, "in learning Logo, the student is not simply solving problems; she is solving problems in a mathematical domain. The objects and processes available to her for the construction of programs are themselves mathematical… the question is whether, in learning to program in Logo, the child may develop a bridge between the pseudo-concrete mathematical world of a computer screen and the abstract world of mathematics" (pg. 336).

There is ample evidence that students can and do transfer the models they build by programming functions into more traditional algebra contexts, but the complexity of those models is limited. In a study of the ISETL programming language, researchers found marked improvement in students' conceptions of algebraic functions (Dubinsky 1992, 1995; Leron and Dubinksy, 1995). Logo has been used to successfully teach students about the notion of angles, measurement (Clements, 1999) and variables (Milner, 1973). While transfer of specific concepts is a promising outcome, this is not sufficient for the kind of higher-level compilation we seek. Will students who learn to program actually perform better than their non-programming peers when confronted with a rich algebra task? Clements (1999) found that students who mastered the concept of variables in Logo were unable to apply that understanding to algebra tasks. In a study that sparked a debate between a young researcher and Seymour Papert himself, Pea and Kurland (1984) found no evidence of improved mathematical reasoning

among students who had been exposed to more than 30 hours of instruction in Logo by tech-savvy teachers in a progressive school. In the very-publicized argument that followed, Pea later questioned the entire proposition that transfer could be expected (1984). SimCalc, on the other hand, was able to demonstrate meaningful transfer into algebra despite having an extraordinarily limited language. What could be preventing this transfer when dealing with more a powerful programming language like Logo or ISETL?

Just as the idiosyncrasies of a program or calculator interface may limit the way students think about algebra, each programming language comes with its own syntax, semantics and restrictions. Students must learn the idiosyncrasies of each new language, which can pose problems for compilation and transfer into the algebraic domain. Like the students from Perkins' (2009) physics classroom, they may draw a faulty distinction between "computer problems" and "math problems," failing to apply what is learned in one domain to the other. Indeed, syntax is often considered the main culprit for transfer, with the implications that students are tripped up by the complexity of symbols, keywords and conventions found in most programming language. A recent study of the programming language Python (Konidari and Louridas, 2010) found that students struggled to understand a number of syntactic rules, despite the language's reputation for being clear and concise. Logo was explicitly designed with a simple syntax, and drag-and-drop languages like ToonTalk, Scratch and ALICE are often held up as

models for teaching children to program. While a simpler syntax has been shown to facilitate *writing code*, there is little evidence to suggest that it also facilitates transfer into algebra. But if syntax is not the limiting factor, something else must be acting as a barrier for transfer into algebra.

In the realm of computer science, the semantics of programming languages is a contentious one. Programmers and researchers alike hold their own preferences and allegiances to one language or another, often because of the way certain types of tasks can be expressed in their language of choice. The fit between language and task is not entirely subjective, however. Because each language has its own semantics, we would expect each language to better suit tasks that can be expressed in those semantics. Scratch, for example, is designed with a focus on tinkering and experimentation (in keeping with its constructionist roots), but this approach has tradeoffs: in addition to demonstrating high engagement levels, the Scratch language has also been linked to poor programming habits (Meerbaum-Salant, Armoni & Ben-Ari, 2011). This tradeoff makes a lot of sense given Scratch's emphasis on engaging younger students, but it may not be desirable for other purposes. Semantics matter, and ideally a programmer would always select the best tool for each job. At this point, we may wish to consider what language one might select if the job was *teaching algebra*: what semantics matter for algebraic tasks?

The Notional Machine

The *notional machine* is the mechanism by which we construct what the computer is doing. Benedict du Boulay (1981) defines it as "the idealized model of the computer implied by the constructs of the programming language." In his study of novice programmers, du Boulay observed them talking about and thinking about their programs as following as set of rules. When the rules did not match the actual behavior of the language, programming errors would appear frequently, reminiscent of Van Lehn's *Mind Bugs*. Linking back to the transfer literature, the notional machine is the mental model that students build when they think about programming. When a new problem is confronted, a student tries to employ the capabilities of the machine to solve it.

In "The black box inside the glass box," du Boulay advocates the use of teaching using notional machines whose behavior is simple and explicit, to facilitate the creation of an accurate mental model. These models display the same phenomena of welding, siloing, and negative transfer (Perkins, 2009). When two languages are similar, a programmer will be able to apply what he or she knows about one language to solve problems in the other. An unfamiliar language may be challenging for a programmer to use *simply because it is different*, not because of any inherent complexity or limitation of the language.

Programming languages differ in a variety of ways, ranging from subtle syntactic distinctions to deep, semantic differences. One of the largest differences

in semantics of various languages is the notion of *state*, in which a language exposes features of the machine on which it is being executed. This concept is deeply embedded in many popular programming languages, such as C, C++, Java, Python, and FORTRAN. Consider the following (annotated) sample code:

```
var foo = 0;          // initialize 'foo' to zero
function f(x){        // declare 'f' as a function of x
     foo = foo+1;     // increment foo
     return foo;      // return the new value of foo
}
var a = f(1);         // store f(1) in a
var b = f(4);         // store f(4) in b
var c = f(4);         // store f(4) AGAIN in c!!
```

The first line of code uses the `var` (variable) keyword to instantiate a new value, `foo`, and initializes it to be zero. The next four lines define a new function `f`, which takes in a variable `x` and performs two operations. These operations *change the value* of `foo`, and then return the `foo`'s new value. The last three lines of code evaluate the function `f` using 1 as the first input, then 4 as the input twice in a row, storing the result in variables `a`, `b`, and `c`. After the whole program is executed, the values of `a`, `b`, and `c` will be 1, 2 and 3, respectively. This result is surprising, since `b` and `c` were both given the result of evaluating `f(4)`. We know from algebra that a function cannot have multiple outputs for the same input, therefore our function `f` is violating the vertical line test! How can this be?

The explanation is that `foo` is not actually a variable, and `f` is not actually a function - at least not in the mathematical sense of the word. Instead, `foo` is *a*

location in memory, somewhere on the computer. We can imagine a box labeled "foo," with a value that is initialized to zero at the start of the program and then changed every time `f` is executed. The programmer must be aware of the value of this box during execution, because the behavior of `f` is not predictable without also knowing the state of the machine. Programming languages that use state in this way, called *imperative languages*, treat a program as a list of instructions for pushing data around inside a machine. The notional machines for these languages can be thought of as having a series of slots, with values being stored, modified and retrieved from them.

However, a stateful notional machine is radically different from the *declarative* machine used in mathematics, in which a function's behavior is defined by the function and nothing else. State mixes the execution of a procedure with its definition, forcing the programmer to consider both the code they write and the machine on which it runs. Mathematicians can rely on the fact that *f(1)* will always evaluate to the same result, regardless of how many times *f* - or any other function - has been evaluated, while imperative programmers have no such guarantee. Translating stateful computation to mathematical evaluation requires a formulation of the stateful machine as an implicit variable, and every function to be redefined over a domain that includes that machine, and maps to a range of possible machine-states. In this construction, *f(x)* would become *f(x, MACHINE)*, and *f* would now be described as $f: MACHINE \times Input \rightarrow MACHINE \times Output$.

While this transformation is possible (and surely educational!), it serves only to raise the barrier for students of algebra. Conflating parameters and procedures with variables and functions is unlikely to help students understanding either one, especially when both concepts behave in such different – and incompatible – ways.

If we want students to build a notional machine that reflects the properties of mathematics, we must ensure that the notional machine gained from programming does not include state. Unfortunately, the use of state is pervasive in imperative programming. When students are confronted with questions about initialization or array indices, they are dealing with state. Common programming constructs like `for` loops and `goto` statements are stateful. Even the familiar `print` command is stateful, since it modifies a machine and has no return value itself. Such functions are often considered to have a "void" return value, indicating that their sole purpose is to change the state of the machine. The simplest program in Java, which is used on the popular AP Computer Science exam, requires students to consider a `main()` function that returns no value whatsoever! State is widely used in microworlds, which are often targeted at children: the Logo turtle and Scratch cat have stateful properties such as orientation and position. Before students have written a single line of code, each of these properties is "initialized" to some value, and students write programs that modify those properties just as our example modified `foo`. When a student wishes to create a variable in Scratch,

they must *assign* an initial value. The notional machine represented by these languages presents obvious barriers for transfer into algebra.

There are, however, programming languages that eschew state altogether. LISP, Scheme, IPL, and ML are all examples of *functional languages*. These languages trace their lineage to the Lambda Calculus, a mathematical language for describing the definition and behavior of functions. Programmers (especially novices) develop habits in relationship to the languages they use. A novice C programmer is likely to use a stateful solution to a problem, regardless of how suitable that solution is to the task at hand. This is problematic for teachers who wish to use imperative programming to solve algebraic problems, as the solutions are likely to involve constructs that inhibit the transfer they seek.

To demonstrate this phenomenon, let us explore an example that is common to both math and programming classes. Consider the stateful (left) v. functional (right) definition to an algebraic definition of the factorial function (middle):

Stateful	Algebraic	Functional

`function factorial(n){`
`  var result = 1;`
`  while(n > 0){`
`  result = result*n;`
`  n = n-1;`
`  }`
`  return result;`
`}`
	$factorial(n) = \begin{Bmatrix} 1, n = 0 \\ n * factorial(n-1) \end{Bmatrix}$	`function factorial(n){`
		`  if(n == 0) 1;`
		`  else n*factorial(n-1);`
		`}`

In the stateful example, the programmer is intimately aware of the machine: a location in memory is named `result` and initialized to `1`; the program uses the

value of `n` as a condition inside a `while` loop, which executes some code until `n` is no longer greater than zero. Inside the loop, the value of `n` is being multiplied by `result`, and then decremented. Finally, the value of `result` is `returned`, as control of the machine is handed back to whatever function called `factorial` in the first place.

By contrast, the functional solution makes no reference to the machine at all. The function is simply defined as a process applied to its input: `factorial(4)` doesn't *return* `4*factorial(3)`, it *is* `4*factorial(3)`. The code in the functional solution is precisely what was described mathematical definition – no more, no less.

It is important to note that both programs will produce the same output for any given input. When treated as a black box, both functions are equivalent. However, their construction is completely different, with one function being an almost-perfect translation from mathematics and the other being heavily grounded in the semantics of a machine executing a program. And while it is possible for each language to construct the other's solution, the semantics of each language make it less intuitive to do so. The imperative `factorial` program describes both a computation and the mechanical process by which it takes place, much as a student might describe long division by including both the underlying math and directives for "move the next digit down next to your result." As with long division, focusing on the mechanical process by which an answer is achieved can

make it easier for students to obtain the answer. However, it may also serve to undermine their mathematical understanding of why it works, as they are overwhelmingly focused on the process instead.

When we consider the process-object gap that students must cross when dealing with functions, the implications of imperative v. functional programming become clear. Imperative languages force students to separate the mathematical object they are programming from the computational process by which it will be evaluated. Given the concrete, experiential nature of programming, one would expect a student's mental model for an imperative language to be flexible and rich – but it would also contain constructs that are incompatible with algebra. In fact, the transfer literature would predict situations in which this model is falsely applied to mathematics, resulting in the sort of persistent "mind bugs" that reflect fundamental gaps or errors in a student's understanding of algebra.

There is ample evidence to suggest that these misunderstanding do in fact occur, and that procedural semantics are a real barrier for transfer into mathematics (Sutherland, 1989; Usiskin, 1998). In interviews with children who had learned to program in Logo, Pea and Kurland (1984) found that children had internalized a *procedural* understanding of control flow, rather than *algebraic* understanding of function application. "Many children believed that placing the name of the executing procedure within that procedure causes execution to 'loop'" (p. 147) rather than to apply that function again to a new set of inputs. Many

students made mistakes that suggested a misunderstanding of simple function application, utilizing only a rigid, context-bound framework for applying common Logo functions like REPEAT (p.147). Similar differences exist in the use of variables across domains (Usiskin, 1998), with students often struggling to generalize a *variable* to stand for more than one value at a time (Knuth, Alibali et al., 2005). The notion that x in the expression "$2x + 5$" could stand for more multiple values (or entire sub-expressions!) is a vital step in symbolizing relationships (p. 69). The fact that a computer happens to execute a computation with one value at a time is merely an implementation detail, but imperative programming requires that students incorporate this detail into their mental model. Transfer is demonstrably possible, but studies like these prove the choice of language semantics to be deeply relevant. If a programming language is chosen as a vehicle for algebra, its semantics should conform to the rules of algebra as closely as possible. At this point, it should come as no surprise that the language of SimCalc is completely functional: there is no notion of control flow, initialization, assignment, or program return. A SimCalc program looks like algebra because it *is* algebra.

Unpicked Fruit

SimCalc opened the door to modeling as an effective optimization strategy for rich algebra tasks, but the range of those tasks is tightly constrained by the limitations of the language and the interface. Full-blown programming languages

have shown positive transfer for a handful of concepts (Clements, 1999; Dubinsky 1992, 1995; Leron and Dubinksy, 1995; Milner, 1973), but evidence of transfer into algebra has remained elusive. It is our contention that one of the primary reasons for this is their pervasive use of *state*. As we have seen in our discussion of the algebra literature, attachment to a process-oriented view of mathematics is a significant obstacle for the understanding of the function concept. With procedural execution so deeply embedded into stateful language languages, it is no surprise that there is a high barrier for transfer into algebra.

The literature on transfer highlights the importance of relevant and rich tasks, making it important that the scope and sequence of any programming curricula be carefully tailored to algebraic activities. This has implications for the choice of programming language, but also for the activities and exercises the students perform. If the ultimate goal is to facilitate transfer into algebra, a programming language should be as syntactically and semantically similar to algebra as possible. Students should work with this language in contexts that they are likely to see in algebra class as well, solving problems or studying properties that are closely tied to algebra. I believe there is unexplored territory for a technology-based intervention that uses functional programming, a simple syntax, and authentic, rich algebra tasks to facilitate compilation and transfer into the algebraic domain.

For my dissertation, I will perform a feasibility study of an intervention called Bootstrap, which uses programming to facilitate student compilation of a rich model for algebraic criteria. Created in 2005, Bootstrap has been refined through repeated trials in afterschool and in-school settings. The intervention uses an algebraically sound subset of a simple, functional programming language called "Racket" (closely related to the well-known "Scheme" language). The lesson plans, handouts and materials confront students with a series of rich tasks, each of which builds towards the design and completion of a working video game. Importantly, the tasks and pedagogy are drawn wherever possible from conventional algebra: the vocabulary, problem solving approach and notation are semantically and syntactically close to recommendations by the NCTM and Common Core Standards. This feasibility study will focus on an in-school implementation across several schools, and include an analysis of student and teacher outcomes.

Chapter 3

The Bootstrap Curriculum

Bootstrap is a 20-hour curricular module that teaches students to program their own video games by completing a series of algebra tasks using a functional programming language. Along the way, students learn the concepts necessary to complete these tasks (order of operations, function composition, linear and piecewise functions, inequalities in the plane, etc.), and apply them to tasks on the computer and on paper. The curriculum is divided into nine units, with each one introducing a new concept. Each unit presents the concept as the solution to a missing piece of their video game, has students apply it to small project, and then asks them to apply it to their own video games.

The idea of having students create a video game is hardly novel. In Bootstrap, the video game merely brings a coherent narrative to the course, and gives students a concrete image of the finished product: a student might lack a sense for which set of inequalities determine a range on the number line, but intuitively knows that a character is onscreen if their x-coordinate falls between two points. The intrinsic motivation to make a video game is carefully channeled, such that each mathematical concept becomes the answer to a question *the student cares about*. In her work with elementary school students, Kafai (1998) challenged students to come up with games that could be used to teach others about fractions. In this activity, the mathematical content was embodied in the mechanics of the

game. By comparison, the mathematical content in Bootstrap is embodied in the *programming process*, with the game being merely an engaging outcome of that process. This more closely resembles Harel's (1990) use of programming activities to build mathematical understanding, though even that work situated the mathematical content in the design of the software, rather than in the construction.

The Shoulders of Giants

Bootstrap's use of functional programming is by no means novel or innovative. Such languages have been used at the university level for decades, and many instructors have developed similar curricula for younger students. One of the most well-known functional programming textbooks is called *Structure and Interpretation of Computer Programs* or "SICP" (Abelson & Sussman, 1985), which was first taught at MIT in 1980 and has been used as an introductory computer science course at universities around the world. The "Wizard Book," as it is affectionately called, uses the Scheme programming language to introduce fundamental computer science concepts, and has been called one of the "Top 9 Books in a Hacker's Bookshelf" (Grokcode.com, 2008). In recent years, however, SICP has fallen out of favor with many instructors. Many teachers felt that the material was too difficult for non-MIT students, earning it an undesirable reputation in some circles as a rite-of-passage for academics. SICP was even removed from MIT's introductory curricula in 2008.

If *SICP* is the academic's bible, *Program by Design* is the working teacher's curriculum kit. The *Program by Design* curriculum (née "How to Design Programs") was first conceived in 1995, and includes multiple textbooks, professional development workshops and software for writing programs. Program by Design represents nearly two decades of work by a community of researchers, professors and educators called PLT (see http://www.racket-lang.org/people.html), and uses a Scheme-like language called *Racket*. The PLT group offered a different explanation for the SICP backlash, (Felleisen, Findler et al., 2004), namely the fact that SICP makes the writing of programs explicit, but not their design (p. 373):

> *While the course briefly explains programming as the definition of some recursive procedures, it does not discuss how programmers determine which procedures are needed or how to organize these procedures. While it explains that programs benefit from functions as first-class values, it does not show how programmers discover the need for this power. While SICP introduces the idea that programs should use abstraction layers, it never mentions how or when programmers should introduce such layers of abstraction. Finally, while the book discusses the pros and cons of stateful modularity versus stream-based modularity, it does so without explaining how to recognize situations in which one is more useful than the other.*

In short, SICP amply teaches the merits of various tools in a programmer's belt, but fails to communicate the design process by which a programmer uses them to go from a problem statement into programmed solution. While never stated explicitly, the expectation is that students will develop the sense for design by working through the examples embedded in SICP.

While not a direct response to SICP, it is no accident that Program by Design includes a sophisticated pedagogical technique for writing programs called the *Design Recipe*, which aims to explicitly address the process a programmer takes to solves a problem. The recipe consists of six steps for writing a function:

1. **Data definition** – Create examples of the function's input and output data.
2. **Contract and Purpose** – Describe the type of data that the function consumes and produces, and writes a concise statement of the function's purpose.
3. **Examples** – Illustrate how several example inputs are transformed into the appropriate outputs.
4. **Template** – Enumerate all of the relevant code that may be used in the function, based on steps 1-3.
5. **Define** – Assembling the needed portions from the template (step 4), define the function.
6. **Test** – Using the examples from Step 3, verify that the function behaves as expected.

The Design Recipe forms the core of the curriculum, which has been implemented successfully at dozens of colleges and universities around the word, as well as a number of high schools. While many programming classes have utilized software that is carefully aligned to the needs of their chosen curriculum, the Design Recipe represents an explicit approach to pedagogy that is also aligned to both the software and the curriculum. It is this emphasis on the trifecta of pedagogy, curriculum and software – and how they must be developed in concert with one another – that has led to my interest in Program by Design. The chief criticisms of Program by Design over the years deal with its emphasis on recursion

and data structures in an introductory class, which are often reserved for upper-level computer science classes and seen as being too difficult for first-year students. At the same time, Program by Design avoids common indexed data types (such as arrays) and the control flow constructs associated with them (such as `for` loops), which are often used in introductory courses which have a history of using imperative programming. These differences have led to concerns that Program by Design is inappropriate for an introductory course. Bootstrap does not address these criticisms directly, as recursion, data structures and arrays are outside the scope of the course. The content Bootstrap leverages from Program by Design is *only* the material one would find in a traditional algebra class.

Introducing Bootstrap

Bootstrap's origin is firmly rooted in Program by Design, and I have had the good fortune to work with many of the PLT team over the last 10 years to guide its development. In many ways, Bootstrap can be thought of as a re-imagining of the first few chapters of *How to Design Programs*, focused on algebra, designed with a traditional K-12 classroom environment in mind, and wrapped around a set of carefully scaffolded projects.

Language and Software

To facilitate knowledge compilation for algebra, a programming language should introduce as few non-algebraic concepts or syntactic rules as possible.

Stateful programming languages require the student to think about the elements of the machine on which their program is running. Tracking attributes like memory, stacks and pointers greatly increases cognitive load, reducing the likelihood of compilation. While PLT and SICP value the simpler semantics of a functional language more for what they *remove* than for what they embody, Bootstrap uses the algebraic semantics of Scheme as the core of a transfer-oriented math intervention. The notional machine for this language, then, is as close to a mental model for algebra as possible.

The Racket language can be thought of as a close cousin to Scheme, bringing the pure-functional semantics to a variety of contexts that are not traditionally treated functionally by other languages. For example, images are treated the same way as any other value, and image-manipulating functions can be composed just as easily as number-producing functions. A math teacher might teach function composition by defining functions *f* and *g*, and having her students compose them in different ways and apply them to different values. Using Racket, these functions can be used to flip, scale, or rotate image values. Instead of figuring out how *f(g(3))* produced the number 18, students instead must discover how an image of a cat was flipped upside down and doubled in size. By allowing the same concept to be expressed using images, Racket offers an engaging alternative to a traditionally dry activity. Racket also offers built-in testing, allowing students to create test cases for their functions (e.g., "if Sally sells no

lemonade today, her income is $0") that are *automatically* compared to the output of the functions. For teachers who are tired of reminding their students to "check their work," this feature offers a compelling reason for students to finally do so ("if you write your test cases, the computer will help find your bugs"). While testing packages can be incorporated into languages like Python and Java, they are far more difficult to use due to the stateful semantics of those languages.

Until 2009, Bootstrap students would download and launch the same software used by Program by Design. However, this approach suffered from a number of logistical and pedagogical problems. First, most public schools lacked the permissions or IT staff needed to download and install software. For the small number of schools where installation was possible, the schools' file-management or backup solutions were either unaware of this new software (ignoring student files during backup) or hostile (erasing student files during the nightly backup). Second, the software also had unexpected impacts on behavior management: to move a student to a different seat, a teacher would have to stop teaching and transfer the files from one computer to another. The time required to do so was unacceptable to most teachers, which forced teachers to keep students together who they might otherwise separate. Finally, Bootstrap diverges from Program by Design in several important ways, and the need to align software to these changes necessitated the creation of an environment that could be tightly coupled to Bootstrap.

The Bootstrap software environment is called WeScheme (Yoo, Schanzer, Krishnamurthi and Fisler, 2011), and is designed to be student-centric. WeScheme lives entirely in a web browser; there is nothing to download or install, and all student files live in the cloud and can be accessed from any device. WeScheme implements a Bootstrap-specific subset of the Racket language, with many additions that are tailored to Bootstrap's audience. WeScheme addresses all of the aforementioned obstacles, and teacher feedback has been extremely positive since the environment was introduced. WeScheme includes a number of student-centered enhancements that go beyond similar products such as Khan Academy, Udacity, etc., such as structured error messages (right) and a stop button that allows a runaway computation to be halted (the technical details of these features are beyond the scope of this paper, but the features themselves are included as specific examples of student-centric design). Finally, WeScheme includes explicit scaffolding for the Design Recipe, in the form of a "Recipe Widget" that presents students with one step at a time, with question prompts and instant feedback.

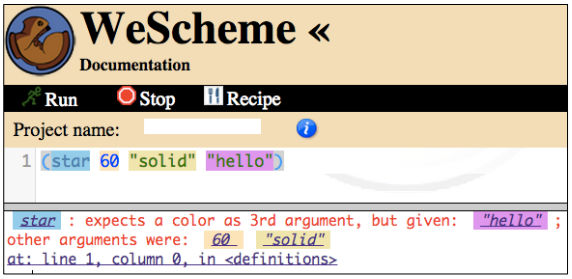

Figure 13 - A structured error message in WeScheme. Note the use of color and readable language.

Pedagogy

Bootstrap introduces a number of pedagogic techniques aimed at various obstacles students face when moving from arithmetic to algebra. The three most

significant techniques are the Circles of Evaluation, the (modified) Design Recipe, and Battling.

Circles of Evaluation

Many teachers of algebra find that their students are still struggling with Order of Operations. The infix notation used by arithmetic is inherently ambiguous, and Order of Operations is a grammatical convention that allows humans to properly parse a sentence like "4+2-7*1." Oftentimes, students practice this grammar by computing the result for a number of arithmetic expressions, with the correctness of their answer serving as a proxy for their mastery of the concept. If a student gets the answer right they are assumed to understand the Order of Operations. This approach suffers from several flaws. First, the assessment itself suffers from false positives and false negatives: a student might answer "1*1+1" correctly without using the proper order at all, and a student who miscalculates "7-5÷4" may have used the right order, but struggled with their long division. More importantly, this approach casts Order of Operations as a *computation* skill, when it more closely resembles a *literacy* skill.

In developing the Bootstrap curriculum, I spent considerable time coming up with something akin to sentence diagramming, but focused on arithmetic. The result of this work is the *Circles of Evaluation*, which is a visual-spatial metaphor for arithmetic that

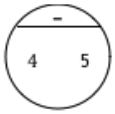

Figure 14 - A Circle of Evaluation for "4 - 5"

involves a set of formal rules for construction and translation. Each circle includes the name of a function and the inputs to that function, and circles can be nested inside one another. Even students who fear anything beyond addition and subtraction feel comfortable in this setting, since it can be introduced using nothing more than those two operations. By offering students a chance to make sense of something they already recognize, there's an immediate sense of accomplishment. Students succeed in the early material because it feels completely within the realm of the familiar, which gives them the confidence to tackle more challenging material. When the material gets harder, students have a set of simple, ironclad rules they can fall back on.

The Circles of Evaluation are a powerful metaphor for introducing functions. Instead of conceptualizing sequences of *operations*, students begin to visualize *functions* being composed on one another. By providing a consistent syntax for function application, students are able to build their understanding of functions atop familiar operations. For example, the Circle of Evaluation would diagram `f(u,v)` the same way as `a+b`, merely substituting *f* for +. This metaphor is easily extended to other functions, from traditional algebraic names like *f* and *g* to those from more traditional programming contexts like `star`, `triangle`, and `sort`.

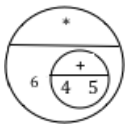

Figure 15 - A Circle of Evaluation for "6 * (4 + 5)"

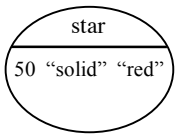

Figure 16 - Circle of evaluation for an image-producing expression

The Circles of Evaluation also serve as a natural bridge to the syntax of the Racket language. To convert any Circle of Evaluation into code, the student "tears open" the Circle, splitting it into two parentheses. The student then copies the contents of the Circle, starting with the function at the top and then reading the inputs from left to right. The Circles of Evaluation dramatically accelerate the speed at which students pick up the Racket programming language, and eliminate many of the most common syntax errors. Both of these are evidence of the technique's impact on cognitive load.

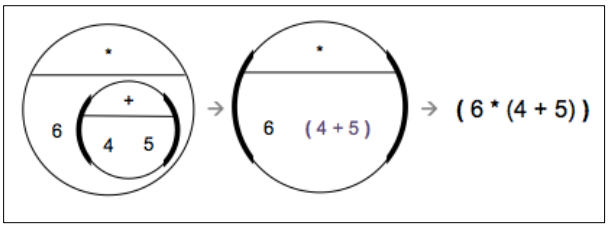

Figure 17 - Converting a Circle of Evaluation into Racket code. The edges of the circles are bolded to show the origin of the parentheses.

The Design Recipe

The six-step PLT Design Recipe is powerful and flexible enough to support a wide range of programming challenges, and is used by upper-level computer science majors at many colleges and universities. This power comes at a price, however, in the form of a substantial cognitive load. Younger students are easily bogged down by the large number of steps, many of which are unnecessary for the types of problems encountered in an algebra class. It is for this reason that Bootstrap uses a modified Design Recipe, which is both simpler and more directly geared towards algebra.

Bootstrap's Recipe consists of only three steps, omitting original steps 1, 4 and 6:

1. **Contract and Purpose** – Describe the type of data that the function consumes and produces, and writes a concise statement of the function's purpose.
2. **Examples** – Illustrate how several example inputs are transformed into the appropriate outputs, and draw a circle around the parts that are *changeable* between those examples. These inputs and outputs are derived from the type information used in step 1, and the changeable portions are given names based on the purpose statement in step 1.
3. **Define** – Define the function, using the Examples from step 2. Everything that remains constant between examples is copied verbatim in the body of the function definition, and the *changeable* are replaced with *variable* names.

The changes from the original Recipe are significant. Note that the Data Definition step from the original recipe has been eliminated, while two of the other steps (4 and 6) are still present in some form. Small portions of the Template (step 4) are embedded in the identification of common structures in the Example step, which masks the separate tasks of identifying concrete examples and abstracting over them. These Examples are then used as automated Tests (step 6), thanks to support from the software environment. Given the simpler class of problems Bootstrap is designed to address, the tradeoff in problem solving power is well worth the gains in cognitive load.

Reducing the cognitive load fosters knowledge compilation, but choosing these three steps has an additional benefit when it comes to teaching algebra. Each

step corresponds to a particular representation of functions: The Contract represents the *domain and range* of a function, the Examples represent a *table of input-output pairs*, and the Definition represents the *symbolic form* of the function. A math teacher might use all three of these representations when explaining a function. These three representations are illustrated below, as they might be written in a traditional math class or Bootstrap class (note the circling and labeling that occurs as part of step 2, in which the changeable elements become variables):

Traditional Math class	Bootstrap class
profit : Integer → Integer each widget (w) sold is worth $5, with $3 in fixed costs profit(**2**) = (5 * **2**) - 3 profit(**9**) = (5 * **9**) – 3 profit(w) = (5 * **w**) - 3	; profit : Integer → Integer ; each widget (w) sold is worth $5, with ; $3 in fixed costs (EXAMPLE (profit **2**) (- (* 5 **2**) 3)) (EXAMPLE (profit **9**) (- (* 5 **9**) 3)) (define (profit **w**) (- (* 5 **w**) 3))

By embodying each representation as a step in the problem solving process, the connection between representations is made more concrete. In addition, each representation is utilized in a different way during the process, explicitly reinforcing the idea that each representation has its own strengths and weaknesses in describing the behavior of a function. The National Council for Teachers of Mathematics and Common Core Standards both recommend that students be exposed to multiple representations of functions, and the Design Recipe provides a concrete framework to do just that. In the context of a math class, the Domain and Range refer to the inputs and outputs of a function machine, while the Examples

are a table of input-output pairs and the definition is the symbolic form of the function.

The Bootstrap curriculum also includes an integrated student workbook, with pages of activities that allow students to practice the Circles of Evaluation and the Design Recipe without using a computer. While some in the CS education field view pencil-and-paper activities as obsolete artifacts to be replaced, the workbook has proven to be incredibly popular among students and teachers. For students, working through the Design Recipe on paper allows them to focus on the task at hand, without the cognitive load of slow typing or the distractions of a computer screen. For teachers, these workbooks are a way to check student work and assign homework to those who may not have a computer at home. Finally, the workbook serves as a valuable bridge between the programming and math domains, reducing the mistaken belief that Bootstrap concepts are only valuable for "computer problems."

To help build this bridge between on-paper math and programming, the workbook pages are filled out by hand, and their contents can be entered directly into WeScheme. However, WeScheme also includes tools to facilitate the use of the Design Recipe directly from within the program. The "Design Recipe Widget" (Figure 18) is a web form

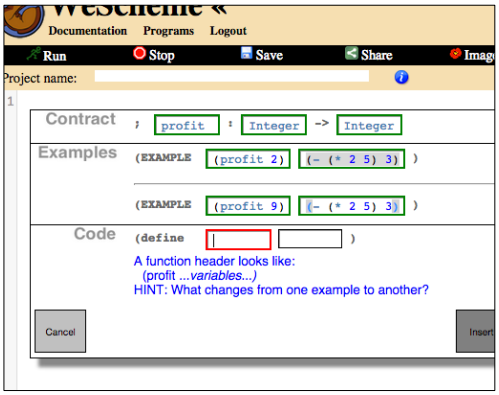

Figure 18 - A partially complete Design Recipe widget in WeScheme.

that matches the pencil-and-paper version, providing limited feedback and error checking to students at each step of the way.

Battles

If *SICP* gives CS majors the tools but no explicit guidance on how to use them, then a similar critique might be leveled against the way problem solving is often taught in an algebra class. Math teachers might spend enormous amounts of time on specific tools and tricks for setting up, simplifying, and solving equations, but then leave it up to students to discover how to use those tools on a battery of word problems. PLT's Design Recipe has made problem solving explicit for CS majors on programming tasks, and Bootstrap's Recipe is intended to do the same thing for middle and high school math students struggling with word problems.

PLT's Recipe allows each step to draw upon the work done in earlier steps, but Bootstrap's Recipe starts out being far more tightly constrained. In fact, students are told that each step must be *completely derived* from the step that came before it. This process is reinforced through a technique called Battling, in which students challenge one another to "defend their reasoning." In a Battle, the Design Recipe becomes the scaffold for the dialog between students, with one student pointing to any item on the page and asking their partner to explain its origin. Their partner must respond by pointing to the location in the previous step from

which that item was derived. A sample battle might unfold as follows (left), for a worked-through Design Recipe (right):

Student 1: Where did you get the name of function?
Student 2: The word problem says to describe the "profit" of the function, so that's what I called it.

Student 1: What about the Domain and Range?
Student 2: The word problem says "number of widgets", and I don't think you can sell half a widget.

Student 1: Why did you circle the 2 and the 9 in your examples, and why did you call them "w"?
Student 2: Those are the only things that change from example to example, and I knew they stood for numbers of widgets sold. I picked "w" because it's a short form of the word "widget".

Student 1: Where did you get the 5 and 3 from in your function definition?
Student 2: The 5 and the 3 were used in both my Examples, so I had to use them as-is in my definition.

Student 1: Where did you get the 5 and 3 from in your Examples?
Student 2: They came from the values in my purpose statement.

```
Word Problem: a company sells
widgets for $5/ea, and spends $3 in
fixed costs. Describe their profit
as a function of the number of
widgets sold.



; profit : Integer → Integer
; each widget sold is worth $5,
with
; $3 in fixed costs
                    w           w
(EXAMPLE (profit 2) (- (* 5 2) 3))
(EXAMPLE (profit 9) (- (* 5 9) 3))

(define  (profit w) (- (* 5 w) 3))
```

The impact of these Battles is threefold. First, they give students a structure through which to talk about solving word problems, and a common language and format for doing so. Second, they reinforce the notion that solutions are *iteratively constructed*, not *spontaneously invented*. Over time, students begin to trust that the process will result in an answer, even if they cannot immediately guess at the answer when reading the word problem. Finally, they improve student confidence and engagement, because students are able to trace their own reasoning in a low-risk, conversational setting.

Teachers often ask students to explain how they arrived at a solution, but that can be extremely difficult even for advanced students, who may be unable to untangle or articulate the process by which they reached a solution. Likewise, it may be impossible for a struggling student to identify *where* they are stuck. Both the struggling and advanced students in these examples are held back by the lack of an explicit problem solving process. The Design Recipe and the Battles aim to be that structure, which a teacher can use with students and those students can use with one another. Now the advanced student can refer to steps of the Recipe to trace their thinking, and a struggling student can articulate specifically where they are stuck.

Curriculum

The curriculum, software and pedagogy are carefully aligned with one another. Bootstrap is a project-based curriculum, inviting pairs of students to brainstorm a simple video game (within a set of constraints) on the very first day of class. Given the near-ubiquity of gaming across gender (Entertainment Software Alliance, 2003), class, and ethnic group for this age group, the video game project sets a concrete goal for the class that is immediately accessible and relevant to students. The fact that the game is something students design for themselves provides a sense of ownership, with students able to visualize "their game" as an incentive for completing the class. The constraints of the game have been carefully chosen so that each element of the project maps directly to a

specific learning outcome from the class. Bootstrap's focus on algebra means that these elements (e.g., animation, key events, collision, etc.) are tied firmly to mathematical concepts (e.g., linear functions, piecewise functions, the distance formula, etc.), with each element of the game introduced as a traditional word problem.

Unit 1

Students discuss the components of their favorite video games, and reverse-engineer a simple game called "NinjaCat" to determine what is changing from one frame of the game to another. In doing so, students discover that the game can be reduced to a series of coordinates, which change in predictable ways. A Dog moving to the left, for example, can be represented as a fixed image with a changing x-coordinate. They then explore coordinates in Cartesian space, using on- and off-screen positions to discover negative values of x and y. For practice, students identify the coordinates for the characters in a game at various points in time. Once they are comfortable with coordinates, they brainstorm their own games and create sample coordinate lists for different points in time in their own game.

Figure 19 - the NinjaCat game

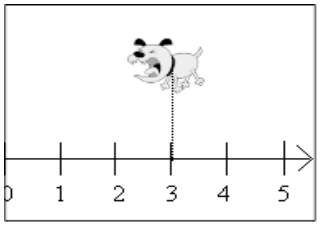

Figure 20 - The Dog's position on a number line

Capitalizing on the high engagement created by the game design activity, teachers introduce the Circles of Evaluation as the first steps in making these games come alive. This motivates a discussion of Order of Operations, and teachers can use any Order of Operations problem from their existing textbooks with the Circles of Evaluation.

Unit 2

Students review the Circles of Evaluation, and the teacher introduces a set-mapping representation for functions, called a *Contract*, in which the function object exists as a means of translating points from a Domain into a Range. Students begin compiling a list of Contracts for every function they can brainstorm. Coupled with their understanding of Circles of Evaluation, students generalize their understanding of functions to include other data types, including Strings and Images. They add Contracts for image-producing functions like `star`, `triangle`, `circle`, as well as image-manipulating functions like `scale` and `rotate`. These functions can be composed to create very sophisticated images, such as flags for various countries. The Circle of Evaluation necessary to generate the flag of Panama, for example is extremely complex, and requires quite a bit of thinking about the coordinate plane. Despite the challenge and rigor of the flag activity, it has proven to be one of the most popular among students and teachers alike.

Unit 3

Students are introduced to an abbreviated form of the Design Recipe (the Purpose Statement is removed to further reduce cognitive load), and are given a series of trivial word problems in order to internalize the pattern of the Recipe. Students define simple functions to generate shapes (e.g., "given a radius, produce a solid red circle of the given size"), and practice using those functions alongside other functions that are built into the language. Students are also shown how to name *values* in the language, the programmatic equivalent of writing "x=3" or "food='pizza'." As a motivation for naming these values, students are shown how to define various images in their video games (Figure 21).

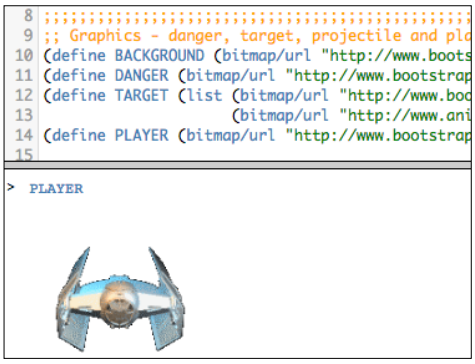

Figure 21 - Definition of game character values

Unit 4

Unit 4 is completely devoted to reviewing and deepening the Design Recipe introduced in Unit 3. Students continue to practice the Design Recipe by applying it to simple problems. In this Unit, students use the full Design Recipe to define a linear function that relates height to time, which is then used to make a rocket blast off (Figure 22). This

Figure 22 - The interactive Rocket activity

is also the point in the curriculum where Battles are first introduced, with the teacher first modeling the exchange and then turning it over to the students to practice with one another. When students complete the rocket activity, they are challenged to make the rocket fly faster, slower, backwards, or even accelerate over time.

Unit 5

Like Unit 4, Unit 5 is devoted entirely to practicing the Design Recipe. This time, however, the word problems students solve are all directly related to the motion of their game characters. After writing a series of linear functions, students see the images they defined in Unit 3 come to life, entering from one side of the screen and flying across to the other. Their excitement is short-lived, however, when they realize that their characters keep moving, leaving the screen and never coming back.

Unit 6

At this point in the curriculum, students are motivated to make their characters come back to the screen once they have gone off the edge. This is used to ground a discussion of what it means for a character to be off the edge of the screen, and how to represent a test for "off-screen-ness" in the form of an inequality.

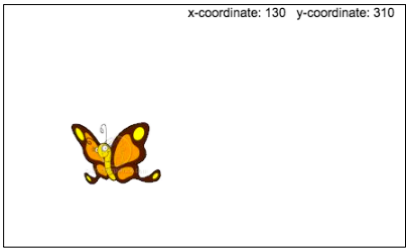

Figure 23 - Sam the Butterfly

Students discover Boolean types, and use them to create programs that test values, and then model scenarios using these programs. Just as the Rocket was used to introduce linear functions, this Unit employs an artifact known as "Sam the Butterfly" to introduce inequalities in the plane (Figure 23). Sam is an 8th-grade butterfly, who has been told to stay inside his mother's yard. Students quickly discover that they can move Sam outside the yard using the arrow keys: being an 8th grader, Sam doesn't always listen to his mom! Students are then informed that Sam might be eaten by an evil, butterfly-hungry cyborg that patrols the neighborhood, and tasked with writing a series of inequalities that will describe all of Sam's safe positions. Students are introduced to `and` (intersection) and `or` (union), and use these functions to compose their inequalities into a single function that returns `true` or `false` if Sam is on the screen. This same code can then be implemented in their video games, allowing their characters to regenerate if they go off-screen.

Unit 7

At this point, students know how to write functions that evaluate all inputs in the same way. This is sufficient for moving a character to the left or right, or for detecting when an x-coordinate is off-screen. However, students quickly realize that this is insufficient for making their players respond differently to key-presses, moving up for one key and down for another. Students once again begin with a

mini-project, this time involving a pizza parlor that needs to produce different prices depending on the topping a customer orders. This unit introduces piecewise functions, which use the Boolean logic from Unit 6 to define sub-domains for a function's inputs. In the pizza parlor example, each sub-domain can be evaluated differently, with students returning one price for a cheese pizza and a different one for pepperoni. Students then use geometry and piecewise functions to add or subtract from their players' y-coordinates, depending on which key the user has pressed. Teachers can challenge students further, by asking them to create "cheat codes" that move their players at different speeds for different keys, cause them to jump to various parts of the screen, or wrap from one side to the other.

Unit 8

At this point, students have an almost-working game. Their characters move around the screen, regenerate when they go off-screen, and their avatars are controlled using keyboard input. However, nothing happens when the avatar collides with an enemy, or when it captures a target. Drawing on their knowledge of how video games work, students are quick to point out that something should happen when their characters are close enough to touch. "Close enough to touch" appears deceptively simple, and many students expect this to be a simple review of

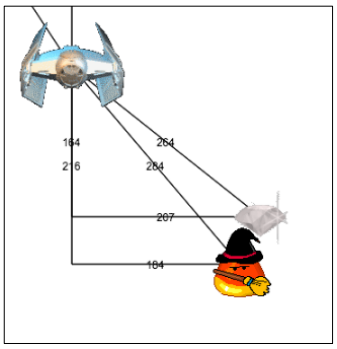

Figure 24 - A running Bootstrap game showing the distance between characters

Boolean inequalities. Students find that measuring the distance between two points is not so easy! The software facilitates this discussion, allowing students to see the distance between each of their characters measured by a series of right triangles (Figure 24). This leads to a discussion and geometric proof of the Pythagorean theorem, with students then programming the distance function into their games. This function can be composed with an inequality to determine collision, which completes the games.

Unit 9

Unit 9 does not include any explicit programming activities, and is instead focused entirely on improving students' ability to talk about their code and design choices. This exercise is taken directly from math activities in which students walk through solutions to a word problem; however, in this case the solutions are presented as Racket code and the walkthrough is scaffolded using the Design Recipe Battles (Figure 25). Students often create tri-fold posters to accompany these talks, mimicking a science fair format. This element of the curriculum is especially popular with math teachers, who see it as chance for their students to show off their work concretely in the same way they might in a science or social studies class.

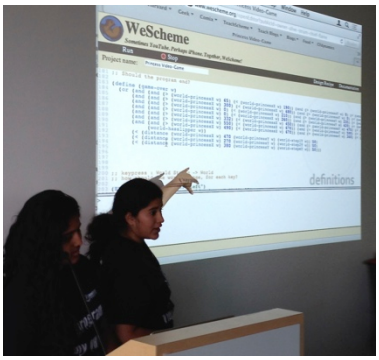

Figure 25 - Bootstrap students present their code to the class

Implementation

The curriculum includes ample supplemental material, consisting of follow-up lessons, homework assignments, warm-up activities, exit slips and quizzes. Some teachers (particularly those using Bootstrap as a stand-alone class) make use of all of these resources, while others use only the core materials.

The core of Bootstrap can be integrated into an existing algebra class or taught as a standalone class. Over 15 hours of the curriculum cover material that is already a part of the Common Core Standards for Mathematics, meaning that a math teacher can integrate Bootstrap into their classroom without having to find 20 additional hours in the school year. The software environment, curriculum, pedagogical techniques and the student workbook are all carefully designed in concert with one another, and are intended to be used together.

Chapter 4

Conceptual Framework

During my survey of the literature, several models were described for the multiple, interconnected representations that make up the "function" concept. Star and Rittle-Johnson (2009) identify the abilities to operate *within* and *between* representations as complimentary: The first skillset involves flexibility when using a particular representation, while the second involves the ability to make connections between more than one. "Understanding in algebra means being able to use multiple representations and then to connect between them" (p. 6). O'Callaghan (1998) provides a measurement for conceptual knowledge that involves four constructs: Modeling, Interpreting, Translating and Reifying (p. 24-26). This task-oriented view of functions draws from several authors, most notably Schwarz & Dreyfus (1995) and Slavit (1997) – both of whom define sets of actions or properties that transcend individual representations. Bloch (2003) also embraced a property-oriented view, defining an understanding of functions in terms of a student's ability to identify properties across representations. Given the emphasis on cross-representational tasks, my framework operationalizes understanding of functions primarily as students' ability to operate *between* them.

This synthesis of the literature into a set of tasks serves as the conceptual framework for the study, which is intended to serve as a meaningful and accurate representation of the *function* concept. The framework, in turn, will guide the

quantitative component of the study and the eventual data collection and analysis. These tasks – and the conceptual knowledge necessary to complete them - are described in a later section. However, it is important to consider the procedural knowledge that is often invoked when working with functions. While this knowledge is valuable, it is not assessed as part of this study.

Procedural Knowledge

Each representation outlined in the literature comes with its own set of basic skills and vocabulary. In order to work with graphs, for example, it may be necessary that students be able to plot points, label axes, etc. O'Callaghan draws a line between these skills and deeper understanding, calling them *procedural,* rather than *conceptual* knowledge (p.22). He also notes that traditional algebra curricula tend to produce stronger procedural outcomes than conceptual ones (Boers-van Oosterum, 1990). In this context, procedural knowledge includes basic familiarity with tables, graphs and Cartesian planes, order of operations, and function application. In my framework, I consider procedural knowledge to be a moderating factor, as these skills are a necessary but insufficient condition for understanding functions.

Conceptual Knowledge

Borrowing from O'Callaghan's (1998) work, I am defining conceptual knowledge in terms of four discrete tasks: modeling, interpreting, translating and

reifying. The Bootstrap curriculum introduces these tasks in the programming domain, using syntax and problem solving approaches drawn from a computer science curriculum. This study examines the degree to which proficiency with these tasks in one domain (programming) can transfer to success on similar tasks in another (algebra).

Modeling

The challenge of many word problems is not in solving an equation, but in setting up the equation in the first place. This activity requires the student to describe quantitative relationships formally, using functions and variables to model the problem. Modeling is described as a high level skill by numerous sources in the literature (Dede, 2007; Edwards, 1998; Herscovics & Linchevsky, 1994; NCTM, 2000), and is well suited to programming based interventions (Resnick, 1994; Wilensky, 1994). A common modeling exercise might describe the flight of a rocket over time, and ask students to derive the symbolic relationship from the written description.

Interpreting

Essentially the opposite of modeling tasks, *interpreting* a function requires students to reason about its behavior, and often to draw connections to real-world situations. This mirrors a similar construct found in Hillel et. al (1992), and Kaput (1987). Interpretation is also highlighted as an essential skill for deep

understanding by standards bodies (MAA, 1991; NCTM, 1989). In contrast to our rocket example, an Interpretation task might provide the student with a height function (e.g., $h(t)=-t^2+10$"), with various questions that can only be answered by reasoning about the meaning of the function ("How high is the rocket after 10 seconds? Is there a maximum height that the rocket will reach?").

Translating

In the literature dealing with multiple representations of algebraic functions, the ability to translate between representations is commonly cited as evidence of deep understanding (McGowen, Demarois & Tall, 2000; Slavit, 1997). In one translation task, a student must convert a function from one representation to another (e.g., "make an input-output table for this function"). In another task, a student is shown a set of possible conversions for a given function and must choose the correct one ("matching"). This latter formulation allows for a multiple-choice instrument, and is closely follows the "linking" tasks employed by Schwarz & Dreyfus (1995).

Reifying

Reification is one of the more esoterically defined terms in the literature. Sfard and Linchevsky (1994) describe reification as "our mind's eye's ability to envision the result of processes as permanent entities in their own right" (p. 194). Slavit (1997) compares the reification of the function concept to understanding the

"bowlness" of a bowl and the "cupness" of a cup (p. 260). Reification can be thought of as a student's ability to view *functions as objects*. These objects have distinct properties, and actions can be performed on them (Breidenbach et al., 1992; Sfard, 2000). These actions may include addition, subtraction or composition of functions (O'Callaghan, 1998, p. 25). Likewise, a student's understanding of variables as more than mere placeholders for a single value requires a rich understanding of functions-as-objects.

Attitudinal Factors

While the primary goal of the curriculum is content-focused, there is justification in the literature for adding an affective assessment to our study. Several authors have found negative correlations between math anxiety and performance (Ashcraft & Faust, 1994; Hembree, 1990), particularly when the tasks are more complex and assessment conditions are stressful. For students taking algebra, these complex tasks are often encountered during high-stakes tests, all but assuring a stressful condition. Standards bodies have highlighted the importance of other attitudes, particularly perceived value and applicability in the real world (NCTM, 1989). Given the potential for these attitudes to impact math performance, and Bootstrap's explicit claims about "real-world" algebra applications, it makes sense to look for attitudinal shifts as part of our study.

While aptitude in computer programming is neither a prerequisite or primary goal of interventions like Bootstrap, programming knowledge may be a

moderating factor for subjects: a student who is completely familiar with programming will be expected to grasp the material much faster than a student with no experience. For this reason, it is appropriate to include programming attitude and experience in our conceptual framework.

Research Questions

In this feasibility study of the Bootstrap curriculum, I will answer the following questions:

1. How do implementations of Bootstrap vary across different teachers? What strategies and practices emerge?
2. What are the major challenges for implementing Bootstrap?
3. How are students' attitudes toward mathematics related to how they experience the Bootstrap curriculum?
4. How is student performance on specific algebraic tasks related to how they experience the Bootstrap curriculum?

Each question will add context to the others: if students demonstrate positive gains in spite of teacher struggles, it may suggest that the curriculum is overly-prescriptive; poor student outcomes in spite of a successful implementation may point to flaws or unrecognized transfer challenges. A mixed-methods approach allows me to add context to raw numbers obtained from tests, and data to back up themes that emerge from interviews.

Chapter 5

Research Methods

Design

Bootstrap has been implemented for audiences outside the target for which it was designed, ranging from a once-a-week after-school program for gifted 4th graders to an intensive, remedial summer class for high school seniors. This study focuses on a convenience sample of six in-school math and computer science teachers, primarily serving students in urban centers in grades 8 and 9. Four of the teachers work in schools where non-white students make up a majority of the student body. One of the six schools is parochial, one is private and the rest operate in the public school system. The teachers range from a first-year instructor to a teacher with more than 10 years' experience. With two exceptions, students at each of these schools score below grade level on state math exams, and in each case the teacher or district selected Bootstrap for its use as a tool to address perceived gaps in students' performance in algebra. While it is not be possible to conduct a full impact study, this research is intended to inform future thinking about student impact in traditionally challenging classroom settings.

Data Set

The data set consists of quantitative pre- and post-test data, measures of implementation fidelity, and transcripts of teacher interviews. I collected this data

from participating teachers and the students in their classrooms during the 2013-2014 academic year. Fidelity measures were collected through a combination of site observations and interviews (30-45 minutes in length), to explore the different ways teachers implement the program and the challenges and best practices for doing so. Site observations were coded using a protocol I have developed (see Appendix D), and interviews were coded to discern common experiences (perceived effectiveness, challenges, teacher confidence and student attitudes). Both the interview and observation protocols were piloted with a second rater to determine their reliability. To detect potential shifts in student performance, I developed a set of pre/post tests that blend measures of core competencies in algebra. These tests underwent cognitive pretesting and reliability studies during the 2011 and 2012 school years, and were found to be reliable measures of the specific competencies I intended to study (see the following section for detail).

A formal impact study is beyond the scope of this dataset, given the sample size and the noise introduced by teacher- and school-level effects. However, having this data for individual classes will allow me to add context to feasibility measures (discussed later in the Analysis section). Through agreements with five of the six schools studied, I obtained permission to access the pre- and post-test data from teachers, each of whom used my instrument as part of the normal course of instruction. While I was unable to obtain IRB clearance for student data from

the sixth school, I was able to include the classroom observations and interview data from the teacher in my analysis.

Measures

Fidelity

To assess program fidelity, I developed an observation protocol (Appendix D) that evaluates the use of the teaching methods and activities that are specified in the curriculum. Observations were taken when the teacher covered Units 4-7, which deal most directly with the concept targeted for transfer (Appendix E). At the end of the year, the same teachers participated in phone interviews, so that observed variations could be further explored. A teacher who chooses an improvised activity over the recommended one, for example, may have done so in response to a particular classroom need or a change in direction. These interviews add context to the variations seen during observation, allowing me to report on what went well, what was challenging, and how students responded.

Affective Outcomes

Student attitudes towards mathematics can have a significant impact on student performance (Ashcroft & Moore, 2009), and are also likely to play a role in academic choices students will make after the intervention (e.g., course selection). I originally planned to use the "Attitudes Towards Mathematics Inventory" (ATMI) assessment in this study, but time constraints required me to

choose between qualitative pre/post measures of affective and transfer outcomes. Given Bootstrap's goal of facilitating transfer, I chose to rely on teacher interviews as a proxy for affective outcomes. My interview protocol includes specific questions about student confidence and anxiety over time, and my analysis will address this outcome in the following chapter.

Learning Outcomes

One of the value propositions of Bootstrap is that students learn to program, making the programming tasks a measure of the one of the program's intended impacts. However, this learning outcome will also have a strong impact on transfer: if students do not learn about functions in the programming domain, they cannot be expected to transfer any understanding into algebra. Short (1-2 min) programming activities (see Appendix B) serve as a form of embedded assessment, to determine if students have learned the requisite programming skills. With the intervention being specifically targeted at teaching programming, it is likely that most students will learn the basic programming constructs of the language. I expect this measure to be extremely positive, though not altogether useful as anything beyond a check on the likelihood of transfer.

Transfer Outcomes

To ensure that the tasks chosen for this measurement accurately represent the field, I developed an instrument that draws tasks from the Massachusetts

Comprehensive Assessment System (MCAS). Massachusetts has adopted the Common Core Standards for Mathematics, raising the likelihood that these tasks are representative of similar assessments in other states that have adopted the Common Core. Given the time constraints of a normal classroom, the questions on the assessment must be carefully targeted to the relevant concepts targeted by the intervention. These state tests cover a wide range of mathematical concepts, and even the items that deal specifically with algebra may not reflect what the literature defines as conceptual knowledge. In my conceptual framework, I laid out Callaghan's (1998) four constructs for appropriately rich tasks for assessing knowledge of functions:

1. **Reification** – Students are given a list of functions, and asked to evaluate the functions' behavior on inputs when composed with one another.
2. **Interpretation** - Students are given a single representation of a function, and asked to explain the behavior of the function in their own words.
3. **Translation** - Students must match representations for a given function (e.g., pairing a list of graphs with the corresponding list of x-y tables).
4. **Modeling** – Students are given word problems that describe a situation, and must define a formula that describes that situation accurately.

This view of functions draws from several authors (notably Sfard, Schwarz & Dreyfus, and Slavit) and is echoed in the Common Core Standards for Mathematics. The questions drawn from these sources are selected for their suitability as tasks conforming to three of the four constructs (see Appendix C). I have chosen to leave out Interpretation tasks, as they are not a focus of the

intervention. To increase the number of items used in the assessment, new questions were formulated to match the wording and category of those drawn from these tests. For a number of the questions, additional steps were inserted into the problem statement to encourage students to "show their work" and to gain better insight into their mental model for functions.

These instruments underwent cognitive pretesting with volunteer students from DC Public Schools to ensure that the wording and presentation was appropriate for the sample population. Pilot studies of three classes (with similar populations to the ones used in this study) found that the composition, translation and modeling instruments had moderate-to-high reliability (α=0.71, α=0.87, and 0.96, respectively). T-tests conducted on the pilot data found student improvement on both the composition (from 0.51 to 0.61, $p \leq 0.01$) and the modeling tasks (from 0.30 to 0.57, $p \leq 0.01$), with no significant change on translation tasks. Given the possibility of both positive and negative transfer, a 2-tailed t-test will be used to analyze transfer outcomes. Despite the small sample size, the use of paired pre- and post-tests should provide sufficient statistical power to detect changes in student performance.

Analysis

The first two research questions (variations and challenges in implementing the curriculum) are addressed through classroom observations and phone

interviews. The observations will detect variation from recommended strategies, informing me of *how* the implementation varies between teachers (RQ #1) and what the challenges were to successful implementation (RQ #2). These variations were discussed in phone interviews, during which teachers shared the challenges, successes and student reactions that explained *why* they chose one strategy over another. This allows me to construct detailed descriptions of what happened in each classroom, including the activities used, challenges faced, and student outcomes.

I also analyzed the interview data for changes in student attitudes towards mathematics (RQ #3). Analysis of the performance data will mirror the analysis used for the anxiety measure: a statistically significant result will allow me to conclude that their performance has shifted during intervention (RQ #4), and I may be able to examine these shifts along gender or achievement lines.

As the creator of the curriculum, I am aware of the risks involved in developing coding categories of my own, which may not align with the experiences of the teachers in the study. It is for this reason that I chose to code the interviews using thematic analysis (Boyatzis, 1998). Thematic analysis uses a two-phase approach. The goal of the first analysis phase is to paint an accurate picture of what was said, before examining that picture for broader themes. These codes are developed iteratively, starting with a schema derived from a small sample of the text and then adapted to larger and larger samples. These codes may be

created, modified or deleted in each iteration, and overly specific codes may be combined and overly general codes may be split into smaller ones. It is important to note that these codes are developed without any eye towards the research questions themselves, limiting the confirmation bias of a deductive approach. While no method can completely mitigate researcher bias, this element makes thematic analysis a particularly effective tool for this study. The codes from the first phase serve as a form of data compression, chunking the interviews into a set of summary fragments. In the second phase, the researcher develops *themes* as a bridge between the research questions and the codes. During analysis, the codes point the researcher towards a location in the text in which a relevant theme is mentioned. When describing a theme's connection to a research question, a closer analysis of relevant fragments is performed to ensure that themes are actually answering the question at hand.

Thematic analysis allows the researcher to make a choice between two levels of analysis when identifying themes. *Semantic themes* are closely tied to the specific sentences or fragments in the text, providing a localized lens into the data at the expense of cross-interview analysis. By contrast, *latent themes* allow the researcher to start with a small set of questions and search the data set for patterns. I opted to employ latent themes, taking a separate pass through the dataset for each of my research questions.

Chapter 6

Results

Fidelity

Bootstrap has been used in various contexts since 2005, and its pedagogical roots in Program by Design go back to the early 1990s. I drew on these experiences to determine which elements of the curriculum were essential, without which a teacher could not be said to be "teaching Bootstrap." The observation protocol (see Appendix D) includes measures for eleven of these essential features, each of which can be scored as "low," "medium" or "high," based on the frequency with which a feature is used. One of the features is based on the classroom environment (students working in pairs, using their workbooks, etc.). There are two features for the *correct* and *consistent* use of appropriate math terminology, given the importance of using mathematical (rather than programming) terms for key concepts. A fourth feature describes the frequency with which a teacher gives students time to wrestle with a problem before walking them through the solution, a practice that is stressed heavily in the Bootstrap training.

Over the years, I have watched many teachers try a new method when explaining a problem, only to abandon it and fall back on a more familiar method when a student needs help. For example, a novice Bootstrap teacher might teach students the Circle of Evaluation when introducing Racket syntax, but then jump

immediately to counting parentheses when diagnosing a syntax error. Another teacher might use the Design Recipe when walking students through a problem relating distance to time, but then go straight back to the word problem when explaining why the variable is called "hours." By contrast, a seasoned Bootstrap teacher would be expected to refer back to the student's Example step and point out that they had used "hours" as a label.

This observation is not unexpected: standing in front of a classroom introducing a problem requires less mastery of the technique than using it flexibly to answer student questions. However, it is essential to use these methods both when introducing a problem and when *assisting* a student who has questions. I developed a set of three "introductory" measures for the Design Recipe (using the Recipe to walk through a problem) and another three "assisting" measures (using the Design Recipe to help a struggling student). For the same reason, I also included a fourth assistive measure, to identify whether teachers fall back on the Circles of Evaluation when students need help with syntax errors.

When calculating a teacher's overall fidelity, each indicator is worth up to 3 points, and the teacher's score is reported as a percentage of the highest possible score (33). When applicable, scores are averaged between two observations. Assistive indicators (shaded) are grouped together, as are introductory indicators (italicized). Travel constraints prevented me from observing Hadiyah's classes, so her scores are absent from the table.

Indicator	Aaron	Ernest	Fatima	Libby	Mallory	*Avg*
Room setup (language table, pair programming, workbooks)	100.00%	83.33%	66.67%	66.67%	33.33%	*70.00%*
Math vocabulary used correctly	100.00%	83.33%	100.00%	100.00%	33.33%	*83.33%*
Math vocabulary used consistently	66.67%	83.33%	100.00%	83.33%	66.67%	*80.00%*
Time for students to work on their own	100.00%	100.00%	66.67%	83.33%	83.33%	*86.67%*
Contracts before Examples	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*	*100.00%*
Examples before Definition	*100.00%*	*100.00%*	*66.67%*	*100.00%*	*83.33%*	*90.00%*
DR used for introducing or reviewing a word problem	*100.00%*	*66.67%*	*66.67%*	*100.00%*	*66.67%*	*80.00%*
Circles of Evaluation used for errors in syntax	50.00%	100.00%	66.67%	100.00%	50.00%	*73.33%*
DR used when students ask for help	100.00%	33.33%	66.67%	66.67%	33.33%	*60.00%*
Explicitly identifying variables in Examples	100.00%	33.33%	66.67%	100.00%	66.67%	*73.33%*
Definition is explicitly tied to Examples	100.00%	66.67%	66.67%	100.00%	33.33%	*73.33%*
Total:	92.42%	77.27%	75.76%	90.91%	59.09%	79.09%

Figure 26 - Fidelity scores for Bootstrap teachers (the four assistive measures are shaded)

Some summary observations about fidelity not covered in the table above:

- Every participant used the student workbooks in their classroom.
- Only Ernest and Aaron displayed the Language Table in their classrooms.
- With the exception of Mallory, teachers were generally successful applying math vocabulary in a programming class.

The heavyweight nature of the Bootstrap pedagogy is evident in this table, with a noticeable drop between introductory and assisting measures. Most participants had high scores when introducing a problem using the Design Recipe (averaging 80% across teachers), or when explicitly asking students to complete the steps of the Recipe in order (100% and 90%, respectively). When a student was stuck using a word problem, however, teachers had a tendency to revert back to non-Bootstrap methods (60%). Teachers also scored lower on measures that identify explicit connections between the steps of the Design Recipe, successfully

"Identifying Variables" and "Connecting the Definition to Examples" only 73% of the time. When a student needed help with a syntax error, teachers used the Circles of Evaluation to diagnose the problem only 73% of the time.

I observed fairly high fidelity from four of the participants, who applied Bootstrap techniques at least 75% of the time. It is interesting to note that Aaron and Libby, who work in more middle-class schools, stood out with scores above 90%. One possible explanation may be that these teachers had more planning time than their counterparts, which is not uncommon in suburban or private schools. Another possibility is that these teachers may have spent less energy on behavior management, allowing them to focus more on internalizing the Bootstrap techniques. However, this may simply be coincidence: Fatima and Ernest work in public schools with traditionally under-performing students, and their fidelity measures were not so far behind (77% and 75%, respectively). Mallory is a noticeable outlier, however, with a much lower fidelity measure (59%) than the other participants. This delta is even more noticeable when comparing her fidelity on assisting indicators.

Teacher Interviews

During the summer of 2014, I conducted video interviews with each of the teachers chosen for this study. Each interview ran for 30 to 45 minutes, and recordings were transcribed and time-stamped for coding using thematic analysis. In discussing my observations for the teacher interviews, I will begin with a broad

description of the coding scheme I developed for analyzing the interviews, before discussing how those codes manifest for each participant.

Codes

The first phase of thematic analysis involves the creation of codes from the raw interview data. These codes are based only on the text itself (not the research questions), and serve as a form of data compression for the bulk of the original text. The initial scheme I developed was based on a reading of the text, starting with the following codes:

- **Student Anxiety** – teacher describes students are described as "afraid," "scared," "worried," etc.
- **Student Interest** – teacher describes students are described as "engaged," "curious," "excited," etc.
- **Student Confidence** – teacher describes students are described as "proud," "certain," "prepared," etc.
- **Teacher Challenges** – teacher describes difficulties (Bootstrap-related or otherwise).
- **Transfer** – teacher identifies specific applications of Bootstrap material to mathematics.

After applying these to a subset of the interviews, I analyzed the remaining, uncoded text and created new codes based on patterns that emerged:

- **Relevance** – teacher describes material as being "useful," "applied," or "personal" for students.
- **Teacher Success** – teacher describes an activity or technique that they feel had a positive impact on his or her students.
- **Collaboration** – teacher describes students as "working together," "helping one another," etc.

- **Step-by-Step** – teacher talks about a process being "broken down," "easy to follow," etc.

I found the Student Interest code to be far too general. Teachers spoke broadly about student interest, citing it as both evidence of success (a positive output) and as a mechanism for further engagement (a positive input). Once I recoded the original subset using the new codes, I found that the code for Relevance had effectively covered the latter use of the Student Interest code. As a result, the definition of that code was constrained to include only those phrases that describe student interest *without* the notion of application or ownership.

The remaining text was then reanalyzed using these nine codes. I enlisted a fellow graduate student to code a subset of the interviews in parallel, in order to gain a measure of inter-rater reliability. She coded two of the interviews (Ernest and Aaron) independently, using the same set of codes I had developed. When comparing these interviews, I found high reliability for six of the codes, nearly all of which were used consistently between readers. Teacher challenge and teacher success were used broadly by both of us, often as a secondary code for fragments that were already coded with something else. Our use of the Relevance code, however, differed in a number of places. Most of these instances saw relevance being used in place of another positive code (Student Interest, Transfer, or Teacher Success). This suggested that the code might be unnecessary, in the presence of the other codes in the schema. I attempted to recode Aaron's interview without it,

but found far too much text that that fell outside of the remaining codes. Given that thematic analysis allows for multiple codes to be assigned to a single fragment, I decided to keep the code for Relevance during my analysis, despite potential inconsistencies in its application.

Participants

Ernest: New York City, New York

Ernest teaches at a technology-focused public school in New York City, and had just begun his first year of teaching when this study was conducted. 72% of the students at Ernest's school are African-American or Latino, and 71% come from families that live below the poverty line. The school has open admissions criteria and does not filter or screen students based on academic background. Ernest became a teacher "after several years of working as a craftsman with a background in liberal arts, mathematics, also history and philosophy." Ernest was assigned to teach a 9th grade class aimed at blending computer science and mathematics. He spent some time tinkering with electronics and programming in his early 20s and used GeoGebra with students during his residency. He had no formal training in programming or computer science, but decided to build a semester-long class around Bootstrap, which he delivered in the fall and again in the spring.

Ernest describes his students as being incredibly varied in their strengths when it comes to math and computer science. “Most students who come into [my school] are not at grade level in their skills,” he says. “I have a large number of students who have never felt successful in a school environment.” Many of his students are interested in technology and talk as if they know something about programming. However, he describes their attitude as “kind of a macho-lingo approach,” pointing out that “they couldn’t even articulate the basic conceptual understanding of a variable as [an] abstraction.”

Ernest found many curricular elements useful, especially the Circles of Evaluation and the flag activity. He felt that there were hard skills being practiced and cited that as the reason he and his students enjoyed it:

> *“…teaching the Circle of Evaluation, it was really a kind of very rapid growth because of being able to really easily pick out misconceptions and ‘This is what you're doing wrong, I need you to fix that.... Some of these Circles of Evaluation got like unbelievably complicated, but they were gorgeous. So as soon as they really kind of got it and could use that heuristic, they were very nice. It's a beautiful, powerful visual.”*

> *“Many of my students that really didn't like computer science remember [Bootstrap] very fondly. They really did feel confident while they were doing it. Since then, what I presented to them was much less well-defined and I can feel a difference and they did, too.”*

Ernest is one of the most reflective teachers with whom I have ever worked. On his own, he studied the entire college-level curriculum from which Bootstrap evolved, gaining a mastery of the programming content that goes well beyond

what is necessary to teach the curriculum. Throughout the year, he expressed frustration that his students were not gaining the level of conceptual understanding he wanted from them. In order to spend more time on the concepts during the spring semester, he taught Bootstrap without the video game component. It did not go well. He explains, "in the second semester, I didn't [use the video game project] and that was a mistake. It was, definitely… the students hated me for it. It felt good for me, but not for them."

The Design Recipe appealed to Ernest on a number of dimensions, calling it "the best curriculum for structured problem solving that I've found." He continues, "Any stop along the way you get stuck, you don't have to worry about it. It's a software development process that we try to teach students, that you can keep working on a project but you can keep trying if the execution didn't work." However, he also found the Recipe cumbersome to teach due to the many steps involved. "The overhead of the pedagogy," Ernest says, "is you have to teach it [and] experience the students struggling, and know why you have to pace it the way you have to pace it and how you have to use the tools, in order to get them to use the tools." Fortunately, Ernest sees this overhead as a one-time investment. He plans to use the Bootstrap materials in the year ahead, noting that he sees opportunities for working the Design Recipe into his practice more smoothly and consistently.

Aaron: Pembroke, Massachusetts

Aaron teaches 7th and 8th grade math at a Massachusetts public school. His is a majority white population, and the students score slightly above the state average (68% v. 52%) on MCAS math test. Just under one-fifth (18%) of the students at his school come from families that fall below the poverty line. Aaron studied Business Management in college and worked for several years in the private sector. He went into teaching for the sense of community and personal satisfaction, and got his certificate to teach mathematics. Aaron has no formal training in programming or computer science. He is certified as a math teacher but spent five years teaching a computer applications class using a mix of Microsoft Office applications and a small amount of Scratch programming. This year, his principal asked him to develop a specialized, math-focused computer science class, which he delivered to his 8th grade students.

He taught Bootstrap as a one-semester elective in the fall and spring semesters, with students spending two of every four days on the material. He describes his students as having serious math anxieties. “They're all very scared,” he says, “students are scared, right off the bat.” Like Ernest’s class, Aaron’s students are excited about technology but have little understanding about what programming is. “They're interested in it, but they don't have much background on what it is or what it could be.”

Aaron found the Circles of Evaluation and the Design Recipe to be very useful, speaking at length about how the Design Recipe gives struggling students the confidence to work through word problems and explain their answers when they're done:

> *"I love the design recipe because it levels the playing field for some kids who are used to really grinding out their work as opposed to students who may skip ahead a little too fast. It forces students to slow down and take a look at their work and really understand what they're doing… It's manageable and they can see the progress from one step to the next and how to start with the word problem and end up with the solution. I love that the students can then explain to you why their answer is their answer. It's never a, 'this is the answer, but I don't know how I got it' answer. They can easily go back and see their trail and what they did. If they make a mistake or if they find that they need to change something, make him go back and everything is right there for him. It's very easy to tweak and adjust to make it work."*

Aaron spoke a great deal about Bootstrap's value proposition as an application of the math students learn in school. "They get that real confidence that what they're learning is not necessarily going to be used later, but it can be used right now," he explains, "The tangible proof is that they have a fully functional video game in front of them." For Aaron's students, the project is what made the concepts from their math classes useful and relevant. Aaron has continued to teach Bootstrap to his students, and plans to train a second teacher at his school this year.

Mallory: Washington, DC

Mallory is the senior technology teacher in the group, having taught various IT and computer science classes for ten years. She currently teaches at a technology-focused high school in Washington, DC. The school is high performing, with 93% of the students passing the districts' standardized math test. It also reflects the demographics of the District, with African-American students making up 92% of the school (Latino students make up an additional 5%) and the majority (59%) qualifying for meal assistance. She studied business administration in college and took a number of business-oriented programming classes out of her own personal interest. As a computer science teacher, she taught classes in C++, Visual Basic, HTML and Scratch, before teaching Bootstrap for the first time in 2011. While the data obtained for this study comes from her high school students, she has also taught the class to 8th graders at a nearby middle school, which has a reputation for having difficult students and poor test scores.

Mallory has been recognized as a master STEM teacher, winning numerous accolades and named "DC STEM Teacher of the Year" *twice* during her tenure. Her teaching style is very positive and upbeat. She praises nearly every student response, emphasizing the success of a correct answer or the bravery and creativity of one that "wasn't quite right." While Ernest spent a lot of his interview time focusing on the conceptual understanding he wanted to see in his students,

Mallory focused more of her attention on affective qualities like comfort, confidence, and ownership:

> *"I truly believe, Emmanuel, that if you can create comfort in the classroom, that's going to open up students to learn and they're going to get confident. Honestly, no matter what, if there's anything I want my kids to have, it's confidence because once they're confident, you can do really anything with them."*

I believe that Mallory's low fidelity score is explained by her accustomed teaching style and background as a pure technology teacher. Mallory struggled with a number of the math terms, at times using a word to refer to the wrong concept (at one point, she referred to "x+y" as a function). Her use of proper math terminology (33%) was far less than that of other participants (83% and above). Mallory also had the lowest score among "assisting" indicators, which may be explained by her deep experience as a programming teacher. After ten years of dealing with syntax errors by finding and fixing them for her students, Melanie struggled to break the habit and instead have her students draw Circles of Evaluation. Finally, her "classroom environment" score was also significantly lower than other teachers, due to her decision to eliminate the use of pair programming.

The project-based aspect of the curriculum was a significant draw for Mallory, for whom students' pride in their finished product is a crucial component of the classroom experience. She appreciated "that students could walk out of there with a video game storyline that they chose." She echoed this sentiment at

several other points in the interview, emphasizing how important it is to for students to create artifacts as part of their learning. “I mean you've got to create, especially when it comes to math.”

This emphasis on pride and ownership led her to deviate significantly in one element of the curriculum, having each of her students work on their own game rather than working in pairs. She saw pair programming as an impediment to ownership, saying, “I have found personally, I want everyone to work out of their video game. It's a serious strategy. It's ownership… What I mean by that is anyone can get up and help anybody whenever they want to. They can help each other, but they all have an artifact at the end.”

Mallory described her challenges at the middle school as being unlike anything she had experienced. The behavior problems she faced were severe and in several of our conversations she described frequent student outbursts as her biggest obstacle when working with the class on earlier material. Despite a number of reservations, however, she decided to try teaching Bootstrap later in the school year. Mallory has enormous faith in her students and believed they would be capable of learning the material and that the behavior management strain would be worthwhile in the end. What she does not expect, however, is to see behavior improve as a result of the increased rigor:

> *“[During Bootstrap], my middle school kids felt very confident and so much so in themselves in an academic manner that their behavior fine-tuned to be more conducive to learn more. They like the feeling*

of being confident. They got very excited because they were very confident… I didn't have any problems with them anymore, and two of them came up to me and said, 'I can't believe I did this and I can't believe you thought I could do this.'"

This year Mallory is teaching the Exploring Computer Science (ECS) curriculum at her high school and has decided to replace the ECS lessons on programming with Bootstrap lessons. She is also returning to teaching Bootstrap at the middle school.

Fatima: Oakland, California

Fatima teaches math at a credit-recovery school in Oakland, which serves students who have fallen off the traditional educational pathway. Several of her students have been expelled or have dropped out of other schools, or have delayed their education due to childbirth or incarceration. The school reflects the surrounding demographics, with 67% of the students identifying as African-American and another 24% as Latino. Her students struggle with math, with *none* of the school's sophomores scoring at or above grade level on the CA math exam. "They're totally scared of math," she explains. "When they come in, they think that it's going to be business as usual, first of all, so they think, 'I failed it so many times, I'm going to fail it again this time.'" However, Fatima is quick to point out the strengths these students possess, and speaks with pride of how she has been able to bring out the best in students with whom others have struggled:

> *"What I like, for example…we had some kids who came to [my school] from that [other] school, and when I tell the principal now how those students turned around, she totally does not believe that those are the same kids that were at her school. Until I showed her the plaque from the robot competition, 'This guy who you kicked out of your school is now on our robot team!'"*

Fatima has been teaching math for seven years and has always known that she wanted to be a teacher: "When I was in the seventh grade I was in a journalism class and we had to write a report about what we wanted to do when we grew up…I wrote that I was going to be a teacher." She has staunchly avoided using programming in her class, despite pressure from her peers. Her aversion to programming is not from lack of confidence, but from a lack of relevance: "For years a co-worker, Mr. Orange, has been trying to get me to try computer science, and things that are related to computer science, but I've always not wanted to do it, because I'm like, 'Where am I going to find the time? How is this related to what I'm doing?'" Bootstrap was the first programming class that she saw as being "real math," and she decided to co-teach it with help from the school science teacher.

As a veteran math teacher in an alternative school, Fatima has seen dozens of alternative approaches to teaching algebra that have left her skeptical of novel approaches. In our conversations in the past, she has characterized herself as someone "with a low tolerance for you-know-what." I was pleased to hear her speak highly of the Design Recipe as a tool for solving math problems:

"[The students] said, 'Oh, this makes a lot of sense.' They were able to write examples for themselves and then to turn them into equations with variables. I'm not one to say the stuff works when it doesn't work, because I have tried some things in algebra that do not work. I've tried cups and chips, and turning things upside down, and equal signs. If it crosses over the line, then it becomes negative. I'm like, 'What are you talking about?' [laughs] I really think the Design Recipe works because the examples make sense. I'm like, 'Just think about what is the cost of three chairs and two pens? OK. Now, let's put some variables in there.' I mean, it's so straightforward. I don't know. I've tried a lot of things with teaching word problems and this, for me, works.

Like the other participants, she also spoke about the importance of applying the mathematical content to problems that her students saw as being personally valuable. In the excerpt below, she compares a word problem from her math book to a similar problem in the Bootstrap curriculum called "Sam the Butterfly":

"Today we're dealing with inequality systems. It's something about if Africa needed some food and they only had up to $500,000, how many packages of rice and how many packages of beans? I'm like, 'Come on now people. Can you give me something real?' How far before Sam gets out the gate, right? That's more realistic than somebody going to drop some packages in Africa because my students can see, 'Oh, man. I need my butterfly to come back.' You should have seen them, too! 'When is it going to come back, Ms. V? It's not coming back. When is it going to come back on the screen?' [chuckles] They were really excited about it, and I was excited about it too. I'm looking forward to doing it again."

Fatima draws an important distinction here between "real" and "relevant" problems. Despite the real-world nature of the conventional word problem (sending rice to Africa), the use of the computer program was able to turn an otherwise irrelevant and abstract problem (trapping Sam) into something that

engaged her students. Many teachers struggle to come up with "real world" applications of each concept. Technology's ability to make narratives more engaging may lower the need to create such problems altogether.

Libby: Jacksonville, Florida

Libby has the strongest programming background of any teacher in the study, having worked as a professional programmer for many years before switching careers. For the last thirteen years, she has taught mostly computer applications in her technology class, with a small amount of programming in HTML and Scratch. She has also been a math teacher for the last three years, teaching 8th grade algebra at a majority white (74%, 19% African-American) parochial school in Jacksonville, Florida. The school does not publish standardized test scores, so it is difficult to gauge how well their students perform relative to others in the district. Libby began teaching algebra the year the study was conducted, but at the time was only able to use Bootstrap as part of a one-semester elective class, which delivered in the fall and spring semesters to a mixed group of 6th, 7th and 8th grade students over roughly 26 contact hours.

Libby spoke highly of the Circles of Evaluation, calling them "one the golden nuggets of the class." Like Ernest, she described the Design Recipe as being high on potential, but wished she had presented it to her students differently the first time around. "Last year I didn't really give the appreciation for it that I

have now... It is elegant, it is comprehensive, it's a wonderful design tool and I think this year they are going to appreciate it a little more, because I do."

Student ownership was also a primary motivator for Libby when using Bootstrap. In addition to the video game, which is constructed in pieces throughout the course, the curriculum includes a number of mini projects that are designed to introduce individual concepts. Libby described these enthusiastically:

> *"I think those are especially good because they get an immediate reward. They define a function, they copy the function into the program and something cool happens. I think those are the kind of things that the kids who aren't really interested in math or...those are the kind of things that will sit with them well, and keep them open to math more than they would be."*

Hadiyah: Chicago, Illinois

Hadiyah studied systems engineering in school but started a career in education as the technology teacher for a parochial school three years ago. Roughly 50% of the students at her school are Asian-American, with the African-American and Latino students making up only 10% and 2% of the population, respectively. While her school does not share the standardized test scores of her students, she expressed the school administration's concern about the math performance of their students. This concern is what led her to adopt Bootstrap, and what convinced her principal to allow the former "business applications" course to be changed.

Continuing the pattern of applications-focused content, her technology class mainly involved word processing, slide-show creation and spreadsheets, with a small amount of Scratch programming or HTML included. Programming is largely absent from what Hadiyah describes as "computer science." Not only does she refer to her applications class this way, she also describes an afterschool "computer science" club in which she "taught another application for photo-editing." Bootstrap was her first experience teaching a formal programming class, and none of her 9th graders had done any programming before entering her classroom.

Like Mallory, Hadiyah took steps to raise the level of personal ownership by having each student work on their own game. However, she then felt frustrated by her students' need to get their graphics to look exactly the way they wanted. "I know it's not a big part of your curriculum, but that was a hard part for them. I didn't feel like I wanted to spend that much time with it because it's like you can use a triangle function. You don't even need this...They had to have, like, their special player according to the way their game looked." I was somewhat surprised by her desire to encourage students to work individually, yet also work together. At times, these sentiments were expressed in the same breath, one after another: "I really wanted to see what they could do on their own. I really encouraged them to share, collaborate, and help each other."

Another parallel between Hadiyah and the other teachers is the statement that Bootstrap requires a significant one-time planning cost.

> *"I would say that that teacher has to have a love of math and be comfortable with it, that teacher has to be a problem solver, and that teacher needs to be ready to spend some time understanding this program, this mode of instruction, and all the components of it. It's like I said before, the second time you teach it, it would be so much easier. It's a lot of up-front work, but it's definitely worth it because you can keep repeating that."*

Themes

The following table summarizes the themes that emerged from each set of codes when looking for answers to my research questions. In most cases, a code is "clustered" around a particular theme, creating a comfortable hierarchy of concepts. For example, the Step-by-Step code appears almost exclusively within the theme for Demystifying Problem Solving, identifying that code as one of the concepts that makes up the theme. In contrast, some codes make an appearance in more than one theme, reflecting the broad impacts of a particular idea (e.g., Student Anxiety) across a number of dimensions. Finally, the Teacher Challenge and Teacher Success codes appeared in every theme, as teachers spoke about their own struggles and triumphs when dealing with student attitudes, outcomes, or classroom implementations. In thematic analysis, a code that cannot be usefully grouped in a subset of themes is often a sign that the code is too general. When I attempted to subdivide the Teacher Success code, I found that different teachers measured their success using existing codes, such as Student Confidence or

Student Interest. I removed the Teacher Success code from the following table, opting to discuss its presence alongside other codes in the following section. The Teacher Challenge code followed a similar pattern, with the exception of its appearance when teachers discussed implementation challenges (the sole use for this code in the table).

Research Question	Theme	Codes
How do implementations of Bootstrap vary across different teachers? What strategies and practices emerge? →	*Content v. Confidence*	• Student Confidence • Collaboration
What are the major challenges for implementing Bootstrap? →	*Start-up Cost*	• Teacher Challenge
How are students' attitudes toward mathematics related to how they experience the Bootstrap curriculum? ↗	*Demystifying Problem-Solving*	• Step-by-Step • Student Confidence • Student Anxiety
How are students' attitudes toward mathematics related to how they experience the Bootstrap curriculum? ↘	*Making Math Important*	• Relevance • Student Interest • Student Confidence
How is student performance on specific algebraic tasks related to how they experience the Bootstrap curriculum? →	*Making Transfer Explicit*	• Transfer • Student Confidence

Table 1 - Latent Themes from interviews with six Bootstrap teachers

Content v. Confidence

When looking for differences in strategy and practice (RQ1), I noticed a theme that dealt with the teachers' own priorities for their students: Some teachers focused most heavily on the *content*, wanting their students to understand the concepts at work and apply them to novel programming and math problems. Others were focused on *confidence*, and wanted to see their students take ownership in their work, or feel more comfortable and empowered. These differences led to variations in implementation, as well as sharp distinctions in what text was coded as Teacher Challenge or Teacher Success. More than 75% of "teacher success" codes for the first group were clustered alongside codes for transfer and step-by-step, with most of the second group's "success" codes being collocated with student collaboration, student interest and student confidence.

Ernest, Aaron, and Fatima (all certified math teachers) were primarily focused on content, wanting to see their students learn how to program and to succeed in traditional math contexts. All three teachers measured their success or struggle through the lens of student mastery. Ernest expressed frustration with his students' grasp of the concepts at work when describing his challenges, and describes his success through anecdotes where students learned specific skills that they could apply elsewhere. Aaron and Fatima spoke proudly when describing instances where students had applied Bootstrap in a traditional math setting.

Mallory, Hadiyah and Libby focused more heavily on student attitudes, emphasizing the importance of creativity, play and ownership. One of Mallory's favorite qualities of Bootstrap is that it "gets the kids to play and manipulate with the math." Bringing creativity to her classroom was valuable for her even in the absence of mastery ("they felt good that they were using math to create their video games, but I don't think it made them experts. It made them comfortable to learn it again"). Hadiyah also emphasized the pride her students felt in the creation of an end product and talked about the video game demo party she organized for her students:

> *"They loved it. They all wanted to play each other's games. In fact, we had a debut or a coming out party, or a launching of games. It was really sweet and nice because a lot of their themes were kid friendly. We had our younger students come in and play the games. It was so neat to see the younger students sitting with the older student and the older student encouraging them on. 'Oh, you got this much points, keep going.' They loved it. They felt really proud of themselves!"*

In talking about her positive feelings from this party, she did not mention evidence of transfer or student learning. Hadiyah's sense of accomplishment came from how her students *felt*, not what she believed they had learned. Libby's reflection on student outcomes were similar:

> *"I think that they kind of feel a pride and an ownership in the video game. I think that will kind of sit with them in a good way. I think it will make them more open to mathematical concepts. It's just a feeling I get, I don't really have any proof of that, but I do think it's going to have some impact in the future."*

The difference between these two groups is striking. One group perceived their success or failure in terms of students' conceptual understanding. What they liked most about Bootstrap was the focus on hard skills and processes for problem solving. They cited concrete examples of mathematical applications as evidence of success, speaking only briefly about students' excitement, creativity or confidence. At the other end of the spectrum are the teachers who judge their success or failure based on their perception of students' feelings. They cited concrete examples of students demonstrating pride, ownership and creativity as evidence of success, and their modifications to the curriculum were designed to enhance these outcomes. This may be an artifact of the divide between math and technology classes, and the personalities of those who are drawn to teach those classes. Math teachers are traditionally given a fixed body of content (e.g., "teach all the content in Algebra 1 by June"), which is structured as a list of testable skills. Technology teachers are given tremendous flexibility in their choice of content, and the lack of standardization makes learning outcomes difficult to measure. It may be no coincidence that Aaron, Ernest and Fatima are all certified math teachers, while Mallory, Libby and Hadiyah teach more open-ended "technology" classes.

Both groups felt that they had learned something from teaching Bootstrap. Ernest reported that he had underestimated the importance of the video game as a tool for student engagement, and Libby found herself pulled towards more concrete forms of assessment. Where she once would allow students to experiment

with small projects over large periods of time, she found herself valuing quick assessments and rapid feedback cycles:

> *"I think keeping to that pace has tightened me up as a teacher, in terms of just assigning a certain amount of time to something and moving on. Just following the lesson plan in that amount of time, the 12 weeks, forced me to go faster than I would like to, but that had a lot of benefits because I saw what kids will produce when you just give them five minutes to produce it. Where in the past, I probably would have given them longer for not much more reward."*

These differences directly impacted the variations in the way teachers implemented the Bootstrap curriculum. Ernest changed his implementation dramatically in the second semester, all but eliminating the video game in favor of additional exercises that he hoped would enhance content knowledge. By contrast, Hadiyah and Mallory jettisoned the pair-programming element of the curriculum and had students work individually, out of a desire to maximize engagement and ownership. This has significant implications for the Bootstrap curriculum, suggesting that different types of teachers may have radically different definitions for success or failure when using the curriculum. These differences result in variations in implementation, teachers' own perception of efficacy and impact, and may influence student outcomes as well. This also casts doubt on the suitability of the "one-size-fits all" professional development that Bootstrap currently offers to teachers of all backgrounds, and hints at the utility of additional scaffolding for each type of teacher.

Startup Cost

A consistent theme across participants was the learning curve involved in using the curriculum for the first time. Bootstrap has a lot of moving parts, uses a number of techniques (Circles of Evaluation, Design Recipe, etc.) that are unfamiliar to teachers and deeply connects two domains (math and programming) when most teachers are only familiar with one. In addition, there are a number of small deliverables, each of which serves as an assessment and plugs into the video game project. All of the teachers described the experience as being difficult, but each one felt that it was a one-time cost.

Having attended only a one-day professional development workshop before teaching, Aaron would have preferred more. "I would say that eight hours would probably not be my minimum," he explains, "maybe three days, on top of that, to really feel like I could really wrap my head around everything. Fatima felt the same way, emphasizing the need to see the techniques in action in order to use them properly:

> *"You need like a week, because you need to be able, as a teacher, to go through the entire 10 units and make your own game from start to finish. Because, you're going to see the bugs that happen, when you are programming and you are going to be able to see, how things are related to one another. You need to know where you are going, because, the students are going to ask, 'Why do we need to do this, and how are these related to the game?'... You definitely need to see the whole curriculum, before you even try to start unit one."*

Nowhere is this theme more evident than in the way teachers talk about the Design Recipe. Every teacher praised its potential and found it useful, but nearly all of them expressed concern about how difficult it was for them to internalize each of the steps involved and communicate it to their students. Libby's students "did OK" following the recipe, but did not really understand why it was important. "Last year I didn't really give the appreciation for it that I have now," she says. Ernest felt the same way, describing the difficulty of using it with his students before fully understanding the power of the tool. "[The Design Recipe] requires the teacher to have taught it before to really understand the power of it and what it can do for students. I know now, but I don't think I did [then]." He articulates the challenge as a classic knowledge-compilation problem:

> *"It takes too long. There's a really long arc between problem and solution…They either can see the answer, and you're just trying to convince them to do this anyway (the whole show-your-work problem), or it's too hard and working through the steps seems like too much. They lack a sense of flow through it. There has to be a way to have all those steps that you can always point back to, so then they're like connected. Say like you do all of it, but at some point it becomes a single motion. I need to get to that stage quicker, where you're having them say, 'I can complete the Design Recipe for this, and I can do it in five minutes'…That's probably what I experienced the most was that they found it tedious and they didn't want to do it and we never got past that point. We didn't break through."*

Ernest points to a not unlike the "chunking" behavior described by the Cognitive Architects. When a task is compiled, it can be executed both faster and more flexibly. Ernest felt that the Design Recipe was too complex for many of his

students' working memory, which suggests a useful area of research for future curricular improvements.

A number of questions in the interview protocol asked teachers about whether or not they made use of additional professional development resources that have been developed in recent years, such as classroom videos, additional exercises, or a teacher discussion board. Despite their difficulties, most teachers made little use of those resources. Coupled with Ernest's feedback, it would be useful to channel the effort spent developing these resources into explicitly identifying when a student has or has not internalized the Design Recipe, and spending more time during the training discussing how to address specific gaps in students' use of the recipe.

Every teacher who expressed concern about the startup cost also felt that it was a one-time price to pay. All six teachers were enthusiastic about teaching Bootstrap again in the future, and felt confident that they would be more successful based on having taught the program in its entirety:

> *Fatima: "[Preparation time] is getting cut in half, at least. Right now, I feel way more comfortable. It's just that initial, right? Because you need to be able to experience everything your students are going to experience, before you even start it. You will need to go through the entire 10 units, before you even think you are going to teach unit one."*

> *Hadiyah: "It's like I said before, the second time you teach, it would be so much easier. It's a lot of up-front work, but it's definitely worth it because you can keep repeating that."*

> *Aaron: "The planning time definitely went down once I was able to experience it for the first time, because I knew I had where the pressure points where I knew I could speed up."*

The findings here point to a number of needed improvements to professional development. For scheduling reasons, school districts often look for a one or two-day training, but the experiences of these teachers suggest that more time would better prepare them for their first time using the curriculum. Bootstrap has recently begun to offer 3-day trainings, which provide opportunities for teachers to practice teaching the material. It would be valuable to interview first-time teachers who attended these trainings and compare their experiences to these participants.

Demystifying Problem Solving

If problem solving is viewed as an innate ability, teachers and students are likely to see it as something that is both difficult to learn and impossible to teach. Looking back to the literature on rich tasks in algebra, problem solving has more to do with selecting *which* process to use in the first place than it does with the actual execution of that process. Word problems exemplify this challenge, presenting students with a verbal description of a situation that must be translated into a mathematical representation. It is no surprise, then that the both Student Anxiety and Teacher Challenge codes are used frequently within this theme.

A Spanish teacher would likely describe their students' struggles in terms of the skills upon which they need to improve. We might expect to hear that their students need to work on using indirect objects, or their use of the subjunctive. It would be a surprise, then, to hear them talk about students' personalities as being "less Spanish-oriented" or lament their lack of "Spanish-style thinking". When asked about the challenging parts of problem solving through math or programming, virtually all of the teachers in the study used language that describes fixed qualities of a student's personality, more than a set of skills that can be taught. Hadiyah "disliked the fact that if [students] are not used to that way of thinking it was hard for them," and Libby felt that only students with certain qualities would enjoy programming ("most students that I find that don't enjoy it are less likely to be detail-oriented, to think logically"). It is more difficult to change a personality than it is to develop a skill, and I was struck by the teachers' choice of words when describing these challenges.

This language echoes the "fixed mindset" described by Carol Dweck (2007), in which a student's ability at a particular task is thought to be unchanging. This stands in contrast to a "growth mindset," in which effort is the vehicle for improved performance. Ernest sees the fixed mindset in his own students, who feel "that you succeed if you're smart. They're a fixed mindset. They're used to thinking that you just have it or you don't." Students in Mallory's class feel the same way ("they either love math or they hate math."), and Aaron has had students

express their fears in terms of their identity as poor mathematicians ("some kids have said, 'I'm no good at math. I'm not going to be able to do this'"). This would certainly explain student fears when it comes to problem solving, and this theme encompasses the majority of the codes for Student Anxiety.

If the mystique of problem solving is a cause for fear and frustration, anything that serves to demystify the skill should be correlated with confidence and success. Bootstrap's focus on the Design Recipe makes it unusual in the K-12 programming space. Rather than emphasizing a list of commands or the creation of a finished product, Bootstrap focuses entirely on a general-purpose approach to problem solving. It was no surprise, then, when codes for Step-by-Step, Student Confidence and Teacher Success showed up together when teachers spoke about the Design Recipe as a tool for teaching problem solving. Aaron described his low-performing students as being afraid of word problems and talked about his high-performing students succeeding without really understanding why. He saw both of these as problematic and felt that the Design Recipe gave both types of students a roadmap for understanding:

> *"I love how with the design recipe you see how you can take apart a complex word problem piece by piece and make something manageable out of it step by step. The kids that really grind it out can totally appreciate it. It's manageable and they can see the progress from one step to the next, how to start with the word problem and end up with the solution. I love that the students can then explain to you why their answer is their answer. It's never a 'this is the answer, but I don't know how I got it' answer. They can easily go back and see their trail and what they did."*

Aaron's description of students being able to "grind it out" speaks to a possible shift from a fixed mindset to a growth one. Ernest wishes that his other classes broke down problem solving in a similar way ("I still look at everything else and want certain things out of it. But what I don't find is systematic problem solving"). Mallory felt successful in having imparted some of the "art" of problem solving. ("We've taught them how to decompose a problem, decompose the solution that we can do these little functions inside"). When problem solving is demystified, it becomes a skill that students can practice and teachers can coach. This quote from Aaron sums up the connection perfectly:

> *"It's okay not to know the final outcome, but resiliency and determination are going to keep you functioning in it…I thought [the Design Recipe] was a really exciting way to learn anything, to be able to fail and see it not work, but understand that it's okay for it not to work and that you can go back and fix it again. That's a hard lesson for a lot of people to do with anything. Students, especially in math say 'If I didn't get it, I'm not going to get it. The ship is sailed. That's it.' But Bootstrap really stresses that it's okay, make it work, and to come back to the tools that you know in order to get something to function in it. It's a powerful critical thinking tool."*

Teachers described the impact of the curriculum, on student confidence. Fatima, who teaches math to some of the most challenging students in the study, explains:

> *"I think it also did raise their math confidence. I have students come into the math class afterwards who were more excited, who would lift their heads higher, definitely."*

Aaron tells a similar story:

> *"The girl who was in a pull-out math class and was selectively mute -- didn't talk unless she felt very comfortable -- was walking around helping other partners tweak their games… She would take time to help them work through the process of what was not working for them…Her math teacher did say -- when she was back in the math classroom – 'Her confidence had changed.' They mentioned that at a meeting. I said, 'Well, I'm going to take credit for that one.'"*

Fatima was the most experienced math teacher in the study, and it was interesting to note that she was surprised by the focus on problem solving:

> *"I think that one thing that I like about Bootstrap -- I don't know if this was one of the designs of it -- but this whole problem solving thing. 'I don't care that you can get the answer. I want to know how you did it and that you can take that and apply it to something else'… The thing about it I like is, OK. First, you write an example. I'm like, 'If I want to figure out how many miles so and so went when they went in the truck. OK, let's just do some examples. How much did you pay if you went 50 miles? How much did you pay if you went 75 miles? OK. What is this thing that is changing? Yes. That's the variable!' You can totally hear my excitement, right? This is how I am in the classroom! When they do that, it changes everything because now it takes this problem from being super abstract to being having some real examples that they can look at. Then, having real examples that they now turn back into the abstract, right? Pretty awesome."*

The fact that she was unsure about whether the curriculum was designed around problem solving is surprising, given how much time was spent discussing this during the training. With every teacher expressing the need for more training time, it is possible that Fatima simply did not have time to internalize the Design Recipe until she actually taught with it.

However, the teachers' own descriptions of problem solving as a fixed quality suggests a different explanation. "Problem solving" is something of a buzzword in education circles, and it may be a phrase teachers hear often enough to drain it of any meaning. If those teachers hold beliefs about problem solving as something that cannot be taught, they may tune out the frequent references to problem solving in the Bootstrap training. If they were not looking at problem solving pedagogy with a critical eye to begin with, it would explain why they only appreciated the connection after seeing the change in their own students.

Making Math Important

Nearly all participants spoke about the importance of math being applied to projects that were both concrete and personally important to students. The fragments coded for Relevance are completely housed within this theme, along with a substantial number of fragments that deal with Student Confidence and Student Interest. Aaron shared the progress his students made when thinking about coordinate axes, describing "students who couldn't quite remember which axis was which totally remembered in the context of a video game of the movement and everything." The game became a draw for students, who now saw math as being a means to an end that they deeply cared about. Aaron describes this phenomenon when teaching the distance formula:

> *"The Pythagorean theorem, something that makes eighth grade students cringe and 7th grade students cry. Immediately, they all*

> *wanted to work and they wanted to understand how it works to make their game work. 8th graders who learned it had said, 'This makes perfect sense. It's the most useful thing I've ever learned in math class. I love this.' The context of the video game and seeing the actual math work, definitely positively increased their confidence in math."*

Fatima talked about how siloed some of the math had become for her, after seven years of teaching. "A squared plus B squared, equals C squared is all I knew, like seriously, as a math teacher…those were just sums that I just had memorized and used them and had no idea." She appreciated Bootstrap's connection between the distance formula and collision detection. Libby, who is now teaching a traditional math class in addition to her technology elective, is using Bootstrap as a way to ground the discussions she has with her students:

> *"I was talking about functions to my seventh graders today and they said, 'Didn't we talk about that in Bootstrap?'...And we talked about the design recipe in the functions and what we use the functions for. We talked about domain and range. We just had a whole concept about, a whole conversation today about functions. I was able to use Bootstrap to make it real for them in terms of the input and the output, the domain and range and what the function did. It was great."*

All but one of the teachers in this study spoke about feeling successful as educators when connecting mathematical concepts to a personally relevant project. It is interesting to note that the exception is Ernest, who was so concerned with students' mastery of the concepts that he actively sought to remove the game from

the curriculum. His interview was the only one of the six that did not contain a single code for "Relevance".

Making Transfer Explicit

This theme divided the participants into two groups, each having different expectations for transfer and different approaches to facilitating it. Both groups observed students making their own, limited connections between programming and math, but only one expected concrete evidence of students applying what they learned in Bootstrap to traditional algebra tasks. The differences between expectations resulted in differences in practice, with one group making explicit connections for their students and the other merely trusting that those connections would happen naturally.

As a baseline, most teachers had anecdotes about students making casual associations between math and Bootstrap:

> *Libby: Definitely. The other math teacher, when she taught the distance formula, the kid said, "We used that in Bootstrap. We figured out how far players were apart and if they should collide."*
>
> *Fatima: Everything that we were learning, the kids even remarked, "Oh my Gosh, you got me into a math class." On the under[hand], it is kind of sneaky.*
>
> *Mallory: For example, when some of my students had Miss F. for algebra 1 and I talked about a function and then we were talking about Circles of Evaluation I heard some students say, "I wonder if we could use this in Miss Fishman's class."*

While it is encouraging to see students make unprompted connections between Bootstrap and traditional mathematics, the literature on Transfer should give us pause. These types of connections are often superficial, and without a teacher explicitly bridging the two domains they are likely to remain that way. Aaron, for example, was surprised to see his students struggle in a traditional math class, instead of applying the Design Recipe they had learned in Bootstrap:

> *"I was teaching another math teacher's class one day while she was out…The kids were staring at the ceiling and staring at the seatwork. They're a couple of kids that have been in my elective class and so I approached them. I said, 'Is everything OK? How's it going?' They're like 'We don't even know where to start.' I was like, 'How do you not know? I mean, the Design Recipe is one of your tools.'"*

Aaron's experience is not surprising. Libby and Mallory both expressed a belief that their students' exposure to Bootstrap would transfer in some way to their math classes, but without having explicitly made the connection themselves they were unsure of exactly *how* the experience would transfer. This is where differences between the groups begin to emerge, with some teachers merely feeling hopeful, and others pushing for evidence of the transfer they expected.

Fortunately, making this connection is not a difficult task. Aaron's anecdote continues, after he explicitly pointed out how the Design Recipe could be used on his students' worksheet:

> *Immediately, the worksheet is done. The kids that were in the Bootstrap elective were finished. Then, they turned around and they*

> *partnered up with other kids. They're like, "Let me show you how this works. Decide what the domain and range is," something that they weren't going to talk about in math class for months. Without prompting, they weren't hesitant to try it in another classroom. As soon as they saw that it works no matter the venue, they were more than eager to use it."*

Once Aaron had demonstrated the use of the Design Recipe for traditional word problems, his students were able to apply it (and were confident enough to teach it to their peers). Libby reported no evidence of transfer when she was teaching the class as a technology elective, but has since led a successful discussion of domain and range with her Algebra 1 students in the current second year. As a veteran math teacher, Fatima made sure to use the Design Recipe with a number of regular math problems and was pleased to see at least one of her students applying it in a regular math class.

> *Fatima: I see one student using the Circle of Evaluation. He was doing some [inaudible] operations. He was just thinking about it in what comes first and stuff. I did see him use the Circle, and I was like, "What?' I do see students using -- they might not call it Design Recipe -- but I know that once I teach students to write examples, I see them doing that like in other word problems.*

Mallory, Hadiyah, Ernest and Libby focused more heavily on the programming aspects of the course, and the excerpts from their interviews that were coded as Transfer range from skeptical to hopeful – but never certain. In contrast, Fatima and Aaron made the connection for their students and spoke confidently about Bootstrap's role in improving students' understanding of math. Libby also found this connection, but only when teaching an actual math class.

Before considering the actual data gained from the pre/post tests, this theme highlights the importance of *teachers' expectations* and demonstrates how this perception can influence practice. While previous themes demonstrated a strong teacher belief that Bootstrap teaches mathematical skills, the parts of the data set coded as "Transfer" only applied to instances where that transfer was made explicit by teachers who expected it, or else referred to very shallow instances of transfer from teachers who did not. This grouping of teachers is similar to the "Content v. Confidence" divide between teachers with a math v. computer science background discussed earlier, with Ernest and Libby switching places. Given the math teachers' focus on content, what could explain Ernest's lowered expectations for transfer? Why would Libby have had higher expectations, despite a background as a technology teacher?

This may point to the role of *course title* as a covariate for teacher background. Ernest, Fatima and Aaron were certified as math teachers and placed a strong emphasis on content. However, Ernest was the only member of that group not explicitly teaching the class as a math intervention. Libby made no explicit connections between Bootstrap and algebra in her technology class, but changed her behavior in her math class. If a teacher's background influences their priorities for student outcomes, the course description may also influence the strategies they employ to achieve those outcomes.

Pre- and Post-Tests

The pre- and post-tests measured three algebraic tasks that map to the constructs selected as part of my Conceptual Framework:

1. **Reification** – students are given the definitions for four functions, and are asked to answer eight questions that compose these functions on various inputs
2. **Translation** – students are shown the graphs of four functions, and must match them to input-output tables
3. **Modeling** – students must answer nine word problems

All questions used on these tests were based on tasks found in the Massachusetts Comprehensive Assessment System (MCAS) tests. The original form of the word problems asked students to select or write down only the solutions, which provides limited granularity for determining where a student went wrong. These word problems were modified slightly to allow students to show their work at each of three stages, with the final one being the solution itself. This was necessary to assess whether the multi-step process introduced in Bootstrap is actually transferred into math. The pre- and post-tests were piloted in the 2011-2012 academic year, and the composition, matching and word problem sections were found to have moderate-to-high reliability (α=0.71, α=0.87, and 0.96, respectively).

All six participating teachers gave their students the pre- and post-tests, paired them for each student, and removed any identifying information. Libby

used the pre- and post-tests with two classes in the fall of 2013, one of which was enrolled in Bootstrap while the other took a different elective. She also administered them to a second class in the spring of 2014. Aaron used the tests with one section of Bootstrap in the fall and another in the spring. Due to the extremely high rate of absenteeism at her school (not unusual for credit recovery schools), Fatimah was able to collect only six matched pairs. Her small sample size lacked the statistical power to demonstrate an effect, so these results were removed from the analysis. While Ernest was able to grade the tests himself, he was unable get permission from his district's IRB to share them with me.

The remaining four participants had enough matched samples for me to perform a matched, two-tailed t-test to determine the effect of Bootstrap on task performance for each of the three tasks (the two-tailed test was appropriate due to the possibility of negative transfer). All of these students took their normal math class alongside their technology class.

Reification

The results in Figure 27 describe the mean scores on function composition tasks for five Bootstrap classes and one comparison class. All eight tasks were framed as a 4-item multiple-choice question, with a correct answer yielding one point and an incorrect or blank answer yielding no points.

Teacher	Sample Size	Composition (Pre)	Composition (Post)	Change
Aaron (fall)	26	2.31	6.85	196.54%*
Aaron (spring)	32	2.56	5.53	116.02%*
Libby (fall)	25	2.00	3.12	56.00%*
Libby (spring)	25	1.80	3.72	106.67%*
Hadiyah	15	4.33	6.53	50.81%
Mallory	14	3.64	3.14	-13.74%
Libby (comp)	26	2.62	3.12	19.08%

Figure 27 - Pre/Post scores for function composition tasks (* denotes $p<0.05$). Comparison group is shaded.

With the exception of Mallory's students, each Bootstrap class showed gains on function composition tasks. All of these gains were statistically significant ($p<0.05$) except for Hadiyah's class (due to sample size). There are two data points here that allow us to decouple the potential effect of the intervention from the effect of the math class. First, Libby's comparison group showed no statistically significant improvement on composition, despite receiving the same instruction in their math class as the experimental group. Second, we see that Aaron's fall semester post-tests were significantly ($p<0.01$) better than the spring semester pre-tests, and there was no significant difference between the pre-test scores for both classes. None of these would be the case if the students' math class had a significant effect on composition tasks, suggesting the increased performance was the direct result of students' experience in Bootstrap.

Translation

In keeping with more conventional cross-representation tasks, the pre- and post-tests included a 4-item matching exercise. Students were asked to match the graphs of four functions to the corresponding input-output tables, for a maximum of four points. The Design Recipe asks students to *create* multiple representations of a single function in the process of solving a word problem, but does not ask them to match these representations. The matching task is the farthest-removed from tasks in the Bootstrap curriculum and utilizes function definitions (syntactically different from Definitions in Racket) and tables (syntactically *and* structurally different from Examples in Racket). The use of a novel task and less-familiar representations make this more of a far-transfer measurement. The results from each course are shown in Figure 28.

Teacher	Sample Size	Matching (Pre)	Matching (Post)	Change
Aaron (fall)	26	n/a	2.69	n/a
Aaron (spring)	32	2.06	3.13	51.94%*
Libby (fall)	25	1.12	1.56	39.29%*
Libby (spring)	25	1.20	2.16	80.00%*
Hadiyah	15	1.13	3.07	171.68%*
Mallory	14	1.21	1.00	-17.36%
Libby (comp)	26	1.19	1.27	6.72%

Figure 28 - Pre/post scores on matching tasks. Comparison group is shaded (* denotes $p<0.005$)

Hadiyah's class, as well as Aaron and Libby's spring classes showed statistically significant improvement ($p<0.005$). Libby's fall and comparison

classes both improved, but their gains were not significant. Mallory's class again showed a decline in scores, though the decline was not significant.

Modeling

The instruments included nine word problems, each of which was derived from problems found on the MCAS. These word problems are typically graded in binary fashion, with the correct function definition yielding a point and an incorrect definition yielding none. Due to the importance of multiple representations and the explicit format of the Design Recipe, two additional steps were added to each word problem, asking students to describe the function in terms of its Domain and Range, and as a series of input-output pairs. This allows for a maximum of 3 points for each problem. The comparison group may also have learned about these representations, but did not use them in the context of the Design Recipe. There was no explicit mention of the Design Recipe whatsoever on the pre or post-test.

If Bootstrap students were able to transfer the steps of the Design Recipe into algebra, one would expect to see their scores increase more than they do for a comparison group. The remaining question, however, is whether this transfer actually results in a greater number of final, correct answers on the algebra test. To answer this question, I created an adjusted score that ignores the additional steps, focusing instead only on the correctness of the final function definition. Figure 29 includes columns for both the raw and adjusted scores.

Teacher	Sample Size	Word Problem (Raw, Pre)	Word Problem (Raw, Post)	Word Problem (Adj, Pre)	Word Problem (Adj, Post)	Change (Adj)
Aaron (fall)	26	1.04	12.65	0.62	4.00	545.16%*
Aaron (spring)	32	3.13	8.09	1.28	3.00	134.38%*
Libby (fall)	25	0.96	5.96	0.80	1.72	115.00%*
Libby (spring)	25	0.92	6.00	0.92	1.96	113.04%*
Hadiyah	15	3.93	18.40	2.00	5.53	176.50%*
Mallory	14	2.36	1.79	1.07	1.07	0.00%
Libby (comp)	26	2.85	1.00	1.54	0.81	-47.40%

Figure 29 - Pre/post scores on word problem tasks (* denotes $p<0.05$)

Every Bootstrap class except Mallory's showed significant ($p<0.05$) improvement on word problem tasks, and this improvement was present for both raw and adjusted scores. The effect size was large enough that even Hadiyah's small sample showed significant gains. Mallory's class saw a decrease in the raw word problem measure, but this gain was not significant and there was no change in the number of word problems they were able to answer correctly. The difference between Aaron's fall and spring semester gains suggests that students in the school's traditional math class may have received additional instruction on word problems during the fall semester.

As expected, the raw scores show far more variation than the adjusted scores, demonstrating the effect of my modifications to the format of the tasks. Not only were students able to transfer the multi-representation process into algebra (as shown by the higher raw scores), they were also able to use this

process to arrive at a larger number of solutions (as shown on the adjusted scores). These gains are even more favorable when held against the comparison group, which saw a *decrease* in performance on word problem tasks. This decrease, while not statistically significant, stands in sharp contrast to the scores of the experimental groups. In keeping with the function composition results, both Aaron and Libby's fall semester post-tests dramatically outscored their spring semester pre-tests, despite being administered at roughly the same point in the school year.

Summary

For the most part, the pre- and post-test results show statistically significant improvement to each of the three tasks used in the study. With the exception of Mallory's class, every Bootstrap class saw evidence of transfer on the modeling instrument, with large and significant gains on their raw scores. Even more notable is that this process, while requiring extra steps and more time from the students, actually resulted in a greater number of correct answers on a time-constrained test. Using a different syntax and a different set of problems, these students were able to transfer a problem solving technique inherited from college CS classes to outperform their peers on traditional algebra problems – without the use of any computer or programming at all.

Since many of the students were also enrolled in a traditional math class at the same time, it is natural to question whether these gains were merely the result of what was learned there, instead of Bootstrap. We can reject the null hypothesis

for two reasons. First, Libby's control group showed no significant gains on *any* of the three instruments, despite the fact that each of these tasks was introduced in their normal math class (with the exception of the raw scores for the modeling instrument). Second, both Aaron and Libby taught two semesters of Bootstrap. We do see a small impact of the math class on pretest scores for the fall and spring semester, in that the spring pretest scores were generally higher than the fall pretests. However, all of the fall *post-test* scores were higher than the spring pretests, suggesting that fall students scored higher on these tasks after a semester of Bootstrap-and-math than the spring students had after a semester of math. If there no transfer had occurred, we would expect to see no significant differences between these scores.

These results highlight the large number of factors at work in education research, including teachers' prior training and experience, the course title itself, and the personal values and belief at work in teacher practice. The teachers who saw themselves as imparting skills valued different things about the curriculum than the teachers who saw themselves as building student confidence, and these differences were evident in both classroom observations and teacher interviews. Aaron (the math teacher) generally saw the largest gains in his students, while Mallory, a veteran technology teacher, had no significant results. It may be that the ways math and technology classes are taught create different expectations and skill sets in their teachers, or that they attract different types of teachers altogether. The

role of pedagogical content knowledge as a mediating factor should not be surprising, and will be an important theme in the following chapter.

Chapter 7

Discussion

When viewed strictly as a computer-programming curriculum, Bootstrap may be weighed purely on its merits as an engaging way to learn certain programming constructs. As a transfer-focused intervention, it must also be evaluated by the degree to which students were able to apply what they learned in the programming domain to traditional algebra tasks. Finally, as a module designed for algebra teachers, it must demonstrate a positive impact on student performance of those tasks.

We can evaluate the first benchmark by looking at evidence of student work. Nearly every student in the study successfully completed the mini projects embedded within the curriculum. With the exception of Ernest's spring semester class (which did not use the video game), the overwhelming majority of Bootstrap students successfully completed the video game project. None of the students had ever programmed in Racket before, very few of them had ever used a text-based language at all. This suggests that Bootstrap students learned enough of the Racket programming language to complete the programming tasks necessary to build the games. These tasks range from simple usage of the graphics API (knowing how to import and manipulate game images) to composing Boolean expressions, programming various linear and piecewise functions, and working with multiple

data types. We may conclude that students acquired some degree of programming in the Racket language, and were exposed to various programming concepts.

The fact that students learned some programming is not surprising, nor is it significant when compared to similar efforts in the field. Where things get interesting, however, is when Bootstrap is examined as a transfer intervention that is designed to teach algebraic content. I will first review these results in the context of the four research questions from Chapter 4, and then discuss broader implications and directions for future research.

Research Questions

How do implementations of Bootstrap vary across different teachers?

Most classes were implemented with fidelity in terms of student artifacts, use of materials, and teaching technique. Every teacher in the study had his or her students complete each Bootstrap lesson in the same order. All but one had their students complete the same product outcomes from each lesson, and a portfolio of student artifacts from one classroom would look nearly identical to a portfolio from another. Every teacher used the student workbooks and WeScheme editing environment. Every teacher in the study used the Circles of Evaluation and the Design Recipe appropriately and consistently when introducing or reviewing material.

There were differences, however, in the *level* to which these teachers adopted these techniques. Several participants used the Design Recipe to introduce a word problem, only to abandon it when assisting a struggling student. This was due more to variations in teachers' comfort with the material than a decision to change the implementation, which each participant felt would have less of an impact when teaching the curriculum a second time. As teachers become more familiar with the Bootstrap material, I expect this fidelity measure to improve.

Working in pairs v. working alone

Pair programming is an integral part of Bootstrap. It is designed to foster student conversations about the material, combat the "lone programmer" stereotype, and to expose students to a practice that has become a standard in much of the industry. Pair programming is used throughout the professional development workshops, where teachers are directed to share a computer and design their projects in pairs, exactly as their students will do. It is also embedded in the Design Recipe "Battles," which provide a scaffold for students to talk to one another about their solutions to word problems. Given the degree to which pair programming is emphasized throughout Bootstrap, it was surprising that two of the six participants decided to have their students work individually. In both cases, the teachers believed that students would be more engaged if they did not have to share their project, and that this would outweigh any potential cost to student

learning. I think it is important to note both of these teachers are Technology or IT teachers, who may be more likely to view their role in terms of student engagement above all else, or may have been influenced by stereotypes of computer scientists as "loners."

Use of the video game as a unifying project

Bootstrap uses the video game to scaffold the course (building the game in concrete pieces), align with algebra content (each chunk is connected to a specific concept), and as a creative, team-based activity. Each concept in the curriculum is first tied to a small project, which presents a singular problem that can be solved only by using the concept at hand. The video game project represents a collection of problems to be solved, allowing students to assemble each algebraic concept into a completed game. This allows for forms of summative and formative evaluation, and gives students a concrete and engaging domain in which to apply the math they have learned. For many teachers, the summative video game project is the highlight of the curriculum. One teacher, however, sought to remove the video game project from Bootstrap altogether. Ernest chose this variation in order to make more time for activities that he felt would drive home the high-level concepts in the curriculum. The end result was a steep drop in student engagement, and Ernest says he would restore the game in future iterations of the course.

What are the major challenges for implementing Bootstrap?

High Startup Cost

Like any novel curriculum, Bootstrap's startup cost was the biggest challenge for the six teachers in this study. Bootstrap's alignment between the software, the curriculum and the pedagogy may be viewed as a strength in some contexts, but the tight connections between these three make for a steep learning curve for new teachers. This suggests a number of potential improvements that can be made to the training itself, such as leaving more time for teachers to practice assisting students, or differentiating the training for technology teachers who may need a refresher on the math content to properly make the transfer connection for their students. Fortunately, all six participants agreed that the implementation challenge is a one-time cost: once a teacher had taught the course once, subsequent implementations did not incur these costs.

Math-heavy content for technology teachers

I initially expected both math and technology teachers to be challenged by course content that was foreign to each. While, neither group described the foreign material as challenging, the fidelity scores tell a different story. Math teachers had little trouble adapting to the programming component, using the programming projects and programming-oriented techniques with high fidelity, while the

technology teachers struggled to use the math terminology in place of the terms and explanations to which they were accustomed.

The tasks used in Bootstrap may also have been skewed in favor of the math teachers. When any of the teachers failed to use Bootstrap techniques to help students with various word problems, they would fall back on their own methods. Based on the transfer outcomes, however, it is possible that the math teachers were falling back on techniques that were better suited to the problems used in Bootstrap. The technology teachers, by comparison, may have found their traditional problem solving methods poorly suited to learning goals of the problems used in Bootstrap.

Being designed as something of a chimera, it is no surprise that both groups saw Bootstrap as familiar. Indeed, this was by design. However, this goal may have created an undesirable impact, in which both groups approach the curriculum as little more than a "language-change" from prior classes. It is not a stretch to imagine an algebra teacher who sees Bootstrap as a math class in which the math looks slightly different, or technology teacher who views it as a programming class with a different language. If this is the case, then Bootstrap's transfer outcomes will always be biased in favor of math teachers, who are already primed to make the connection to algebra.

Viewing problem solving as a teachable skill

Being able to describe a skill is a prerequisite for teaching a student to master it. I believe that each of the teachers in this study understands the importance of problem solving, but lacked the means to articulate the process itself as being separate from a student being "able to get the right answer." When their students needed help, teachers fell back on unstructured approaches (repeating the method that had already been unsuccessful, asking the student where they were stuck, telling them to ask a friend for help, etc.). Ultimately, it appeared that teachers began the study with an unstated expectation that problem solving skills would form if students *could only solve enough problems*. It is worth noting that this observation is similar to one of the critiques leveled at the *SICP* curriculum, in the computer science domain (Felleisen, Findler et al., 2004).

Teachers' response to the Design Recipe was extremely positive. All six praised the power of the Recipe for introducing a problem, and spoke to the potential of the tool for assisting students who needed help. Many teachers, however, felt that they only began to realize this potential after working with the pedagogy for some time. This divide was confirmed by the fidelity scores, which showed teachers correctly and consistently using the Design Recipe to introduce problems, but struggling to use it when students ran into trouble. All of the teachers felt that they understood it far better after teaching with it, and reflected

on habits they wanted to change or develop for their second time through the curriculum.

Making transfer explicit

Based on the established best practices for facilitating transfer, each teacher was given information about how important it is to make the connection between programming and math. This was communicated to teachers in the Bootstrap materials themselves, and in the PD workshops where such connections were modeled, discussed and dissected. In practice, however, it was far easier for math teachers to make this connection for their students than it was for the technology teachers. There are two factors that could account for this discrepancy, dealing with teacher expectations and teacher ability. Math teachers may have made these connections simply because they had higher expectations of transfer than their counterparts. In the interviews, the technology teachers spoke proudly of affective changes in their students, which they believed would make their students more confident or comfortable *in the future*. By contrast, the math teachers were satisfied when they saw students apply the concepts they learned in Bootstrap to conventional math tasks *during the course*. Both groups of teachers felt they had met their goals, but they clearly started with different goals. The other possible factor is the ability of teachers to make these connections in the first place. Given that technology teachers may be less familiar with the target domain (e.g.,

Mallory's fidelity scores for using math terms), they may have felt less comfortable making these connections.

How are students' attitudes toward mathematics related to how they experience the Bootstrap curriculum?

Due to the absence of student-level outcomes for this measure, I can only report on teachers' *perception* of student attitudes. Each of the six participants listed confidence, relevance and enjoyment as affective challenges they faced when working with students on math content. All six teachers described their students as being afraid of algebra tasks, viewing them as being disconnected from their own interests and/or being dry and boring to complete.

Confidence comes from Transfer

Teachers spoke highly of the Design Recipe's influence on student confidence. One immediate takeaway was the Recipe's role as a road map, giving every student a consistent first step. Instead of staring at a blank page not knowing where to start, students could always begin by asking themselves for the Name, Domain and Range of the function. After working through more of the material, however, teachers also saw an impact in the way students related to problem solving itself. As they began to trust that each step would get them to the next one, students became more confident that they would arrive at the answer even when it was not immediately apparent.

There was a notable difference between the language teachers used to describe changes in their students' confidence levels, depending on whether or not they explicitly asked their students to apply what they had learned to a conventional paper-and-pencil algebra task. The math teachers who made this connection with their students spoke about their students' confidence levels entirely in the present tense. In contrast, the other teachers spoke in terms of future changes, believing that the intervention "will be helpful down the road," or that students would be "more open to [a math concept] in the future." In the absence of student-level measures, however, this distinction is purely speculative.

"Relevant" doesn't mean "Real"

Teachers spoke about the challenge of creating math problems that are relevant to students. One avenue to relevance is to come up with "real world" problems, demonstrating a concept's connection to the physical world around us. Traditional math books are full of problems like these, involving trains leaving Chicago, ladders leaning against brick walls, or pennies falling from towers. Unfortunately, having a "real" problem is no guarantee of relevance. Fatimah described a problem involving food-aid for Africans, a real world application of linear functions that her students nevertheless found completely irrelevant.

By tying each concept to a portion of their video game, Bootstrap immediately makes them relevant to students. Trapping a butterfly or responding

to key-presses may not be "real" in the physical sense, but they are problems that the students readily tackle in service to a creative goal. This relevance improved recall as well, as students were able to link a concept to a tangible experience. In the previous chapter, Aaron described how students who couldn't remember the difference between the x- and y-axes had no trouble at all in the context of a video game, and how excited they were to learn about the Pythagorean theorem in order to make their characters collide.

The Pythagorean theorem did not change just because it was programmed into a computer, and the method used to introduce it in Bootstrap is found in many conventional algebra textbooks. The change Aaron describes is due to the context into which the theorem is placed, in which the distance formula becomes the solution to a problem that matters to his student. Programming allows for the creation of authentic, mathematical programs that are completely disconnected from the physical world, but feel relevant and important to students.

The game is key for engagement

Several teachers commented on how much their students enjoyed the smaller projects, which are used at each level of course. Whether it was making a rocket fly, trapping a butterfly or drawing flags, teachers reported high levels of enjoyment from their students. However, the video game project stood out above all others as a central theme for students' enjoyment of the program. At the end of

the course, teachers who had implemented the video game project reported very high engagement from their students, as they showed off their finished products to one another or to an audience of parents and teachers. The game itself seems to play a pivotal role in student enjoyment: as mentioned above, Ernest saw a steep drop in his students' enjoyment of the program when he removed the game component, despite keeping the smaller projects.

How is student performance on specific algebraic tasks related to how they experience the Bootstrap curriculum?

High fidelity implementations showed gains

Pre/post data was obtained from five classes. When presented with a conventional, paper-and-pencil algebra assessment, four of the five showed gains on reification (50+%), translation (40+%), and modeling tasks (113+%). With the exception of Hadiyah's class scores on tests of function composition (due to sample size), every one of these gains was statistically significant. Given Bootstrap's focus on word problems, the gains on modeling tasks are particularly important, and deserve further discussion.

The Design Recipe is the three-step process with which all programming problems are solved in Bootstrap. The modeling instrument asks students to solve traditional algebra word problems, and prompts them for three representations that correspond to the three steps in the Design Recipe. The third step is the answer

that students would be asked to provide on a standardized test, and scores on this instrument were compared in raw (all three steps) and adjusted (just the answer) form. The first two steps are not commonly used in algebra 1, so one would expect gains on these steps only for students who received the intervention. The raw measurement is purely for the purposes of determining how well Bootstrap students were able to transfer the Design Recipe itself into the algebra domain, not to compare how well they performed relative to a control group. With the exception of Mallory's class, all Bootstrap classes showed large and statistically significant gains in their raw scores. This suggests that students learned to execute the steps of the Recipe and were able to transfer that understanding into the target domain of algebra. These results were obtained using a different syntax (standard algebra notation instead of Racket) and without the use of a computer, using a series of conventional algebra tasks drawn from state standardized tests, and without any explicit reference to the Design Recipe. These gains were consistent across income levels, urban and rural schools and public and private schools.

Transfer requires domain knowledge

Both Mallory's class and the control class had slightly negative performance changes across the three measures, though none of these changes were statistically significant. Mallory scored much lower than other teachers on fidelity measures, specifically those that deal with math vocabulary and deep use

of the pedagogy to solve math problems. All of Mallory's students completed the programming projects used in the curriculum, and proudly presented them to their peers. Clearly they learned the programming content, but failed to transfer this into the math domain. Based on the transfer literature in chapter 2, the lack of transfer outcomes is likely the result of Mallory's long history as a technology teacher, which may have made it more difficult to draw explicit and accurate math connections using proper terminology.

Limitations

The narrow scope of this study leaves many stones unturned. Since my hypothesis regarding transfer is focused on only a small selection of rich tasks, the instrument represents only a small fraction of those on a comprehensive algebra exam (though there may be unexpected impact on other tasks). Given the limited scope of the experiment, it is not possible to explore the differences between radically different delivery models (e.g., implementations as an afterschool program vs. a conventional classroom). Constraints on gathering data have forced me to exclude the race and SES status as a dimension in the data, and the limited fidelity data may be masking unknown variations in implementation and test conditions. Finally, while the small sample size allows us to explore potential effects on the students chosen for the study, it does not allow us to make claims about the program's effect on a wider population. A larger study with thousands of

students would enable a more rigorous analysis, using hierarchical models to account for teacher- and school-level effects.

Due to tight constraints on the length of the assessments used, this study was also limited in the type of effects that could be analyzed. I had originally hoped to use the 40-question Attitudes Towards Mathematics Inventory to gauge affective outcomes, rather than relying on secondhand teacher reports. Future work could include an affective study using this or related measures. Given the current interest in the field of programming education, it would also be valuable to repeat the study using a summative assessment of specific programming outcomes.

Implications

Even with the caveats discussed in the prior section, the findings in Chapter 6 provide encouraging evidence that transfer from programming into algebra is possible. As a feasibility study, the emphasis here has been entirely focused on the intervention. The variation in transfer outcomes across teachers, however, highlights the importance of the person *delivering* the intervention. This variation also appears in teacher interviews and fidelity measures, which found significant differences in the ways that teachers conceptualize student learning, the expectations they hold for their students, and the degree to which they were able to internalize the problem solving process used in Bootstrap. It should come as no surprise that teachers' knowledge plays a role in student outcomes, but this data

implies three specific things about *what teachers should know*, in order to teach algebra through computer programming.

Prior work has addressed questions about "what teachers should know" separately with respect to the use of technology and the teaching of mathematics. Shulman (1986) describes *pedagogical content knowledge* (PCK) as "the most useful forms of representation…the most powerful analogies, illustrations, examples, explanations and demonstrations – in a word, the ways of representing…the subject that make it comprehensible to others" (p. 9). This is distinct from mere *content knowledge*, which is the ability to carry out the tasks in a particular discipline. Simply put, the knowledge required to *teach* something is not the same as the knowledge required to *do* that thing.

In what is essentially "PCK for technology," Mishra and Koehler (2014) identify the knowledge necessary to use software programs, devices, and programming in the service of teaching. They call this construct *technological pedagogical content knowledge* (T-PCK), and describe the ways in which this skill differs from simply being adept with various technologies. A math teacher might know how to use a calculator, but be unable to use that calculator to explore student conjectures or address a misconception. The latter use case is a clear example of T-PCK. Related work has extended this definition to include the use of specific programming constructs (Ioannou & Angeli, 2013).

In the domain of mathematics, PCK is described as the mathematical understanding that teachers must have in order to best explain concepts to students, and to dissect and correct student misconceptions of those concepts (Ball & Bass, 2000, Thompson & Thompson, 1996). Recent work in the math education field has attempted to map PCK alongside a number of other constructs (e.g., content knowledge, knowledge of the curriculum, etc.). This knowledge, as shown in Figure 30, is summarized as *mathematical knowledge for teaching* (MKT). In a quantitative analysis of MKT's effect on student outcomes, Hill, Rowan and Ball (2008) established a measure for MKT, which was then successfully used as a predictor of student achievement outcomes for first and third grade students.

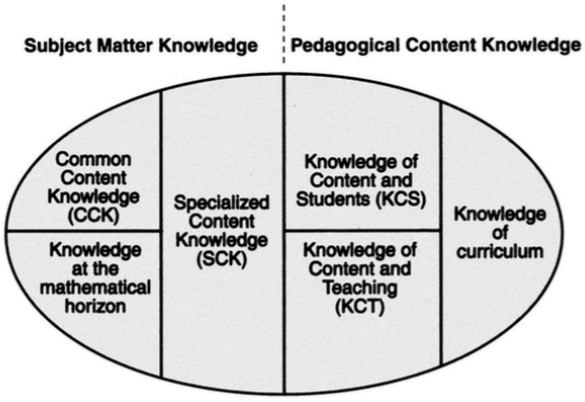

Figure 30 - Domain map for mathematical knowledge for teaching (Hill et al., 2008)

While the scope of this study was designed to answer questions about a specific intervention, the results from the previous chapter hold implications for T-PCK, PCK and MKT: for educators or policymakers who seek to improve algebra achievement by way of computer programming, these three skillsets must be considered together.

A programming intervention must be designed with T-PCK in mind

For Bootstrap, the engineering effort required to make the WeScheme programming environment suitable for algebra tasks was significant, but the tool itself did little to make the math relevant for students. Ernest's removal of the game from his spring semester class serves as an comparison case: all the technological bells and whistles were there, but student engagement suffered without a carefully selected project to tie it all together.

Teachers have always known that a fun, creative project is a powerful tool for engagement. Devising a rich task that can be carefully broken down into math-aligned components is the result of careful curricular design and software support for that design. Keeping the algebra tasks authentic throughout the process was a necessity so that students would maintain their confidence when the computer was taken away, and that requires a programming language and a vocabulary that closely mirrors that of traditional algebra. Algebra tasks do not necessarily become more engaging because they are presented on a computer. What a computer does is allow for abstract, algebraic tasks to become the *means to an end* that students care about. The effective use of the computer, in this case, was accomplished through a curriculum developed with T-PCK in mind.

It is tempting to believe that moving a task from paper to the screen is enough to make that task engaging, and increase learning outcomes. I believe this notion places emphasis on the wrong part of any technology-based intervention.

Simply put, *it is not about the technology*. Digital flashcards are just as engaging as regular flashcards, and graphing a line on a computer is no more exciting than doing so on paper. The value proposition of a technology-based intervention is not the technology's ability to represent existing activities. Rather, it is the technology's ability to make new activities possible that can make content more relevant or engaging, and to support a pedagogy that builds confidence in students. Future work in this field should consider not just the benefits of engaging technology, but the T-PCK necessary to use that technology to produce positive achievement outcomes. Policymakers who invest in programming classes for schools should be sure to consider the resources and time necessary to develop teachers' T-PCK for programming.

Tools and curricula must support mathematical PCK

Having students teach a computer to carry out a series of computational steps has been shown to help them internalize those steps, and studies have demonstrated transfer from programming into some of these tasks (e.g., computing with fractions, constructing geometric figures). Unfortunately, transfer from programming into algebra has largely eluded researchers. If functions represent a shift from computing answers to solving problems, it is possible that an intervention aimed at algebra must focus less on the sequence of steps involved in *running* a program, and more on the problem solving aspect of *writing that program* in the first place.

In recent years, computer science has been linked to a set of skills known as Computational Thinking (CT) (Wing, 2006). The notion that writing programs teaches transferable problem solving skills is popular, with Wing's claim that "computational thinking is a fundamental skill for everyone, not just for computer scientists" (pg. 33) finding ample agreement within the field. In Chapter 1, I described the long-held belief that algebra is a natural domain for this sort of transfer (Feurzeig, W., Papert, S. et al. 1970), and the more general idea that programming teaches math has been around for decades (Khan, 1996; Papert, 1972; Resnick, 2009). If computational thinking teaches transferable problem solving skills that are a natural fit for algebra, then one would expect to see significant transfer between programming and algebra. However, the lack of transfer evidence calls this claim into question.

In Chapter 2, I described a number of specific transfer barriers between the two domains. Imperative programming presents a notional machine that is conceptually incompatible with mathematics. Imperative languages are extremely popular among K-12 teachers, but their reliance on state makes them a poor choice for learning algebra. Meanwhile, the current trend in graphical, block-based programming makes it easier for students to write code without syntax errors, but syntactic precision is a necessary component of an algebra class. Avoiding syntax errors altogether may not be the right tradeoff if the goal is transfer into a domain where symbolic precision is important. Finally, few (if any) of the most popular

tools and languages offer built-in support for writing and testing examples, which are crucial components of mathematical problem solving. Debuggers that allow programmers to inspect the values stored in memory may be valuable parts of T-PCK, but have no proper mapping to mathematics (which lacks any notion of assignment). It is important to note that many programming tools and interventions have demonstrated positive outcomes in terms of student engagement and have powerful learning outcomes on various programming constructs. I would expect, for example, that students who build a video game in Python or Scratch would learn a great deal about any number of computational thinking constructs. But even excellent, thoughtfully designed programming interventions may be completely unsuitable for math instruction, if they embody constructs that are foreign to (or actively work against) mathematical PCK.

Bootstrap's transfer outcomes are the result of more than the incorporation of T-PCK. The WeScheme environment was constructed to support a functional programming language with deep support for common pedagogical techniques drawn from mathematics. The curriculum was designed from the ground up to facilitate transfer into algebra, using mathematical terminology, best practices from the field of math education, and projects designed around specific algebraic concepts. While the previous section's implication is that tools and curricula must be developed alongside T-PCK if they are to be successful in teaching computational thinking, the second implication is that they must also explicitly

support mathematical PCK to be successful at *transferring* that thinking into mathematics. I believe that Bootstrap's support for algebraic PCK is what makes the transfer outcomes possible for teachers in this study. The differences between those outcomes across teachers, however, make an even stronger claim about what teachers should know to facilitate that transfer.

Teachers should possess MKT

PCK for mathematics is just one component of Mathematical Knowledge for Teaching, standing alongside knowledge of the curriculum, content, and others. Recent studies have conceptualized measures for MKT, and identified the construct as a strong predictor of student outcomes (Hill et al., 2005). If MKT is shown to be a meaningful factor in math instruction, it should come as no surprise that it is a meaningful factor in math *transfer* as well.

The Design Recipe, which conceptualizes problem solving in the programming domain, is a sophisticated pedagogical tool for expressing ideas and challenging student assumptions. As such, we may use teachers' adoption of the Design Recipe as a proxy for their T-PCK of the programming domain. Bootstrap adapts the Design Recipe to the constraints of an algebra class, effectively placing it inside the intersection of T-PCK for programming and PCK for mathematics. I assumed that Math and Technology teachers would have similar challenges and successes when using the Recipe, situated as it was equally between their two domains. However, I was surprised to find significant differences in the way each

group approached the Design Recipe – differences which played out in the ways they implemented the course, and saw their own role as educators. These coincided with differences in the degree to which their students were able to transfer what they learned into the domain of algebra.

Two of the programming teachers eliminated the pair programming aspect of the curriculum, convinced that working in teams would reduce student engagement and dilute the affective outcomes they sought. All of the programming teachers made far fewer explicit connections to mathematics, struggled to deeply apply the problem solving pedagogy, and reported feeling hopeful and optimistic that the class would help students in the future. The math teachers made many more explicit connections to mathematics, had higher fidelity when using the Design Recipe, and reported *seeing* student understanding of one or more mathematical concepts improve. This by no means implies that the programming teachers did a poor job, or that they failed to achieve a valuable set of learning outcomes. The programming teachers in this study are talented, accomplished, successful teachers. It is worth noting that all of the teachers in this study reported that their students learned to program, and reported high levels of engagement when using the video game project.

The results of this study suggest that algebraic transfer demands more than the use of mathematical PCK: successful transfer from programming into algebra may also demand some amount of MKT. While this study did not evaluate the

MKT of participating teachers, it is reasonable to believe that those with years of experience teaching mathematics are likely to possess greater MKT than those who did not. It would be valuable to explore the impact of MKT on Bootstrap teachers' performance in future work.

Conclusion

When implemented with fidelity, the Bootstrap classes in this study were able to demonstrate significant gains in traditional algebra tasks after a programming intervention based on a different syntax, a unique set of programming problems, and a problem solving approach adapted from a college-level computer science course. The demonstration of positive transfer from programming into algebra is exciting, but the findings should not be generalized to suggest that merely "learning to program" would result in mathematical gains for students. Bootstrap's success rests on a conscious selection of programming language, software tools, curriculum and pedagogical practice that are drawn from the algebraic domain. This selection likely assumes a certain degree of MKT on the part of the teacher, which puts additional constraints on how and where the program can be implemented.

The notion that programming can be used in the context of an algebra class - by an algebra teacher – suggests that math teachers have a valuable role to play in current efforts to bring computer science to schools. Algebra's position as a required class for all students makes it particularly appealing from an equity

standpoint, compared to self-selecting classes like electives and after-school programs. Schools that are struggling to find room in the schedule and staff to teach a programming class may be able to solve both of these problems by integrating programming into their regular math classes, and improve their math outcomes in the process.

Finally, this study may be thought of as a useful footnote to the assertion that Computational Thinking teaches transferable skills. Transfer between computer programming classes and other subjects is possible, but the language, tools, curriculum and pedagogy of those programming classes must be carefully aligned to those subjects and the skills required to teach them. While most researchers and practitioners would agree that this alignment is important, the history of programming and transfer suggests that the *degree* of alignment required is extraordinarily high. The implications of the study raise this bar even higher, pointing to the impact of mathematical knowledge on transfer outcomes.

Appendices

Appendix A: Observation Protocol

Teacher:	Rater:	Date:	Lesson #:

Indicator	Fidelity
Room setup: Teacher has students writing in Bootstrap workbooks, working in pairs, and has a Language Table posted someplace visible.	☐ Workbooks Used ☐ Pair Programming ☐ Language Table Displayed
Teacher uses algebraic vocabulary correctly when discussing relevant concepts	☐ Minimal ☐ Sufficient ☐ Exemplary
Teacher uses algebraic vocabulary consistently when discussing relevant concepts	☐ Minimal ☐ Sufficient ☐ Exemplary
Teacher refers students back to the Circle of Evaluation or Contracts when assisting in syntax errors	☐ Minimal ☐ Sufficient ☐ Exemplary
Teacher makes explicit use of the Design Recipe when introducing or reviewing Word Problems	☐ Minimal ☐ Sufficient ☐ Exemplary
Students are given time to work on their own	☐ Minimal ☐ Sufficient ☐ Exemplary
(Only applicable in Units 4-9) Teacher refers to the Design Recipe when assisting students who ask for help	☐ Minimal ☐ Sufficient ☐ Exemplary
(Only applicable in Units 4-9) Step 1: Students are directed to complete the Contract for a function, before giving examples or defining it	☐ Minimal ☐ Sufficient ☐ Exemplary
(Only applicable in Units 4-9) Step 2a: Students are directed to give Examples for a function, before defining a it	☐ Minimal ☐ Sufficient ☐ Exemplary
(Only applicable in Units 4-9) Step 2b: Students are directed to identify variables for a function, before defining a it	☐ Minimal ☐ Sufficient ☐ Exemplary
(Only applicable in Units 4-9) Step 3: Students define a function by explicitly referring back to their examples and/or contract	☐ Minimal ☐ Sufficient ☐ Exemplary

Appendix B: Examples of formative programming assessments

`f(x) = x + 3`	`(define (f x) (* x 17))`
What is the name of the function? a. `f` b. `x` c. `3` What is the name of the variable? a. `f` b. `x` c. `3` What is `f(2)`? a. `6` b. `5` c. `x`	What is the name of the function? a. `x` b. `17` c. `f` What is the name of the variable? a. `x` b. `17` c. `f` What is (`f 1`)? a. `17` b. `117` c. `x`

(+ 1 4) will be _____	(> 0 5) will be _____
(/ 4 2) will be _____	(= 1 9) will be _____
(- 0 9) will be _____	(<= 2 2) will be _____
(string-length "bat") will be _____	(string=? "dog" "cat") will be _____

Appendix C: Transfer outcomes (pretest)

All the questions on this page refer to the following four functions:

f(x) = x +1 **x(f) = f^2**

g(y)=2y-5 **p(u, v) = 2u+3v**

1. What is the value of f(2)? (circle one)

0 1 2 3

2. What is the value of x(5)? (circle one)

10 25 36 Can't be evaluated

3. What is the value of g(0)? (circle one)

0 -5 15 25

4. What is the value of p(1, 2)? (circle one)

21+32 8 3 22+31

5. What is the value of f(2+3)? (circle one)

3 4 5 6

6. What is the value of g(f(3))? (circle one)

3 9 11 Can't be evaluated

7. What is the value of x(f(2))? (circle one)

9 16 25 Can't be evaluated

8. What is the value of p(1, f(3))? (circle one)

10 12 14 16

9. Match each of the formulas below with the corresponding table. (One of the matches has been done for you.)

$m(n) = n^2 - 10$

n	m(n)
10	56
11	60
12	64

$m(n) = n^2 - 2n$

n	m(n)
-4	16
-6	36
-8	64

$m(n) = 8 - 3n$

n	m(n)
0	-10
1	-9
2	-6

$m(n) = n^2$

n	m(n)
5	15
6	24
7	35

$m(n) = 4n + 16$

n	m(n)
2	2
4	-4
6	-10

10. The label on the can of paint that Chang bought stated that 1 gallon of paint will cover 300 square feet. The function *feet*(g) represents the number of square feet that g gallons will cover.

a. *What are the domain and range of feet?*

domain : _____ range: _____

b. *Fill in the output column for the function feet(g), completing the two examples provided to show how the number of square feet that can be painted relates to the number of gallons provided.*

f(2)	
f(3)	

c. *Write the function* `feet(g),` *that represents the number of feet that g gallons will cover.*

feet(g) =

11. The total for a phone bill, *t*(m), starts at $19, plus an additional $0.25 per minute *m* of use.

a. *What are the domain and range of t?*

domain : _____ range: _____

b. *Make a table for the function t(m), that shows how the total bill is related to the number of minutes of use.*

c. *Which of the following equations can be used to determine the total monthly bill, t, for m minutes of use? (circle one)*

t(m) = 0.25m + 19 *t*(m)=0.25m – 19 *t*(m) = 19m + 0.25 *t*(m) = 19m - 0.25

12. The table below shows a relationship between values of *x* and *f(x)*:

x	1	2	3	4	5	…
***f*(x)**	3	6	11	18	27	…

a. *What are the domain and range of f?*

domain : _____ range: _____

b. *Fill in the output column for the function f(x), completing the two examples provided.*

f(4)	
f(6)	

c. *Which of the following equations describes the relationship between x and f(x) in the table? (circle one)*

f(x) = 3x *f*(x) = 5x-2 *f*(x) = x^2+2 *f*(x) = x^3

13. Ashley studied for one hour less than twice as many hours as Melissa studied. Let *m* stand for the number of hours Melissa studied. The function *a*(m) represents the number of hours Ashley studied.

a. *What are the domain and range of a?*

domain : _____ range: _____

b. *Make a table for the function a(m), that shows how the number of hours Ashley studied is related to the number of hours that Melissa studied.*

c. *Which of the following equations describes the relationship between m and a(m)? (circle one)*

a(m) = ½m - 1 *a*(m) = 1 – ½m *a*(m) = 1 – 2m *a*(m) = 2m-1

14. A university has 6 times as many students as professors. Write a function *p*(s) that describes the number of professors in relation to the number of students *s*.

a. *What are the domain and range of p?*

domain : _____ range: _____

b. *Fill in the output column for the function p(s), completing the two examples provided to show how the number of professors is related to the number of students at the university.*

p(60)	
p(180)	

c. *Write the function* `p(s),` *which represents the number of professors at a university with* s *students.*

p(s) =

15. Laila is having shirts made with a logo printed on them to promote her band. The total cost is a one-time fee of $75 to have the logo designed, plus $8 per shirt to print the logo. Write an equation that Laila can use to determine the total cost *C*(x), in dollars, to make *x* shirts.

a. *What are the domain and range of C?*

domain : _____ range: _____

b. *Make a table for the function C(x), that shows how the cost is related to the number of shirts printed.*

c. *Write the function* `C(x),` *that represents the cost to make x shirts.*

C(x) =

16. Ms. Gleason is opening a new restaurant. She has enough booths to seat up to 40 people, and is ordering tables to fill the rest of the seating space. Each of these tables can seat up to 6 people. If *t* represents the number of tables Ms. Gleason orders, write a function *p*(t), which shows the number of people *p* that can be seated at booths and tables.

a. What are the domain and range of p?

domain : _____ range: _____

b. Make a table for the function p(t), that shows how the number of tables is related to the number of people that can be seated at the restaurant.

c. Write the function `p(t),` *that represents the number of people that can be seated at tables and booths.*

$p(t) =$

17. Jeff completed a hiking trail in *t* hours. Michelle completed the trail in half the time it took Jeff to complete it. A function m(*t*) represents the time it took Michelle to complete the trail compared to Jeff.

a. What are the domain and range of m?

domain : _____ range: _____

b. Fill in the output column for the function m(t), completing the two examples provided to show how the number of hours it took Michelle compared to Jeff.

m(10)	
m(20)	

c. Which of the following equations describes the relationship between t and m*(t)? (circle one)*

$m(t) = 2 \times t$ $m(t) = 2 \div t$ $m(t) = t - 2$ $m(t) = t \div 2$

18. There are twice as many cats at a pet store as there are dogs. Write a function *d*(c), which describes the number of dogs based on how many cats *c* there are.

a. What are the domain and range of d?

domain : _____ range: _____

b. Make a table for the function d(c), that shows how the number of dogs is related to the number of cats at the pet store.

c. Write the function `d(c),` *which represents the number of dogs at a pet store with* c *cats.*

$d(c) =$

Appendix D: Interview Protocol

Introduction (~1min)

Hello! Thanks for taking the time today to talk about your classroom experience with Bootstrap. We're trying to learn how teachers use the curriculum, what brought them to it in the first place, and how it can be improved in the future.

I'll be asking a little about your own experiences with teaching and programming before Bootstrap, as well as your experience using Bootstrap with your students. You can ask me questions at any time, and you don't need to talk about anything that makes you uncomfortable. This interview will last up to an hour, but you can stop the interview at any time.

There are no "right" or "wrong" answers to any of the questions here – I'm hoping to get the most accurate sense for how things went in *your* classroom, from *your* perspective. This is meant to be more of a conversation than anything else, so that I can learn from your experiences and improve the program for other teachers.

Before we get started, do you have any questions?

Teacher Background (~5min)

1. Tell me a little about your background. How did you come to be a [math/CS/science/etc] teacher?	Teachers' prior experiences with math and programming may influence their perception of Bootstrap, and their choices when implementing it. (RQ1-2)
2. How long have you been teaching [that subject]?	(see above)
3. *(Time permitting)* Have you used programming in your classroom before? a. If so, what did you use? What did you like/dislike about using programming with your students? b. How did your students respond to programming? c. Did you find the programming to be a useful learning activity? Why or why not?	(see above)
4. Tell me about your students. Who are they? a. How confident or excited do they feel about math? How do they feel about using computers? b. What do your students struggle with the most? c. How much programming experience have your students done before they come to you?	Prompt teachers for their own *perception* of their students. This will help address RQ3-4, in which teachers report changes in student attitudes and performance.

Planning and Preparation (~5-10min)

1. *Before* the start of the school year, how much time did you spend on Bootstrap-related	Primarily addresses RQ2 (challenges for implementation), but informs RQ1 as well (different strategies).

planning?	
2. *(Time permitting)* Before the start of the school year, how did you intend to integrate Bootstrap? (Self-contained module? Integrated throughout the class?)	While not directly connected to a specific RQ, this attempts to get at a possible disconnect between teachers' expectations prior to the start of the class.
3. What was the hardest thing about integrating Bootstrap into your curriculum?	RQ2 (challenges for implementation)
4. Did you receive any in-person professional development prior to the start of the school year? If so, what was your experience there?	While not directly connected to a specific RQ, this may allow for some depth of analysis when talking about challenges. If it turns out that teachers who received little or no PD struggled more than those who were trained, we can draw some conclusions about the types of challenges faced by teachers.
5. *During* the school year, how did you wind up integrating Bootstrap?	RQ1 (variations in implementation)
6. During the school year, how much time did you spend on Bootstrap-related planning?	RQ2 (challenges for implementation)

Teaching Practice (~10-15min)

1. Was it difficult to gain access to computers for the programming portions of the class?	RQ2 (challenges for implementation)
2. Which, if any, of the following resources did you make use of this year: lesson plans, student workbooks, homework assignments, warm-up activities, exit slips, YouTube videos, or teacher discussion board?	This question gets at a few related issues: (1) did the teachers know about these resources? and (2) were these resources helpful in overcoming the challenges they faced during implementation?
3. Tell me about your experience introducing the Circles of Evaluation. How did students react? Did you find this to be a useful teaching tool?	RQ3-4 (student performance and attitude)
4. Tell me about your experience teaching the Design Recipe. How did students react? Did you find this to be a useful teaching tool?	RQ3-4 (student performance and attitude)
5. Did you have your students work in pairs? Why or why not?	RQ1 (variations in implementation)
6. If a student got stuck on a problem, how often did you use the Circles of Evaluation or the Design Recipe to help them? If you used them, can you tell me a little about *how* you used them?	RQ1-2 (variations and challenges for implementation). This also speaks to how much teachers were able to internalize the pedagogical techniques used in the curriculum.
7. Did you find yourself using the Circles of Evaluation or the Design Recipe during periods where you were not expressly	While not directly connected to a RQ, this allows us to get at how well teachers were able to blend the Bootstrap material into their traditional instruction. I

teaching a Bootstrap lesson? *(If so, ask for examples…)*	expect answers to this question to be deeply connected to answers about RQ2 (challenges).
8. What did you like about teaching Bootstrap?	*Not directly connected to a RQ, but valuable for the purposes of describing teacher experiences.*
9. How (if at all) do you think that teaching Bootstrap a few times might change the amount of effort required from a teacher?	Addresses RQ2 (challenges in implementation), and may help to tease apart challenges that are inherent to the curriculum from those that are a function of a standard learning curve.

Student Impact (~5-10min)

1. How did your students respond to the use of a video game as a class-wide project? *(Going deeper: Did you see an impact on student engagement? Student learning?)*	Loosely addresses RQ3-4 (changes in attitude and performance). The goal here is to attempt to separate the role of *video game* project from the various pedagogical techniques used in the curriculum.
2. Did you see your students using any of the Bootstrap material when working on things that were not expressly part of Bootstrap? *(If so, ask for examples…)*	Hints at how well students are able to transfer what they've learned between Bootstrap and traditional algebra tasks.
3. How would you describe Bootstrap's impact on your students'… a. Interest in programming? b. Confidence in programming? c. Interest in mathematics? d. Confidence in mathematics?	RQ3-4 (changes in attitude and performance)
4. (Time permitting) Can you speak to the role of the various mini-projects (the flying rocket, sam-the-butterfly or luigi's pizza) on student engagement? *(Going deeper: Did you find yourself referring back to those when discussing related concepts during traditional instruction?)*	This gets at two of the issues raised above – (1) the role of projects, the ability for the teacher to internalize the curriculum and (2) use it to facilitate transfer into their traditional classroom practice.

Goals (~3min)

1. What, if anything, do your students get from Bootstrap?	RQ3-4 (changes in attitude and performance)
2. How, if at all, has Bootstrap changed your teaching practice?	RQ1-2 (variations and challenges for implementation)

References

Abelson, H. and Sussman, G. J. et al. (1985). *The Structure and Interpretation of Computer Programs*. Cambridge: M.I.T. Press and New York: McGraw-Hill.

Abelson, H. and DiSessa. A. (1986). *Turtle Geometry*. The MIT Press. Cambridge, MA.

Anderson, J.R. (1983). *The Architecture of Cognition*. Harvard University Press. Cambridge, MA.

Anderson, J.R., Corbett, A. T., Koedinger, K. R. & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*. *4*(2). pp. 167-207

Anderson, J.R. (1992). Automaticity and the ACT* Theory. *The American Journal of Psychology*, *105*(2). pp. 165-180

Ashcroft, M.H. and Moore, A.M. (2009). Mathematics Anxiety and the Affective Drop of Performance. *Journal of Psychoeducational Assessment*. *27*(3). pp. 197-205

Ausubel, D.P. (1968). Educational Psychology: a Cognitive View. (London, Holt Rinehart and Winston).

Ball, D. L., & Bass, H. (2000). Interweaving content and pedagogy in teaching and learning to teach: Knowing and using mathematics. In J. Boaler, *Multiple perspectives on the teaching and learning of mathematics* (pp. 83–104). Westport, CT: Ablex

Balacheff and Kaput (1996). Computer-Based Learning Environments in Mathematics. In Bishop, A., Clements, K., et al. (Eds.) International Handbook of Mathematics Education. (pp. 469-501)

Berger, M. (1998). Graphic calculators: an interpretative framework. *For the Learning of Mathematics*. *18*(2), pp. 13-20

Bloch, I. (2003). Teaching functions in a graphic milieu: What forms of knowledge enable students to conjecture and prove? *Educational Studies in Mathematics*. *52*(1), pp. 3-28

Blume, G. W., & Heckman, D. S. (1997). What do students know about algebra and functions? In P. A. Kenney & E. A. Silver (Eds.), Results from the sixth mathematics assessment (pp. 225-277). Reston, VA: National Council of Teachers of Mathematics.

Booth, L. (1989). Seeing the Pattern: Approaches to Algebra. *Australian Mathematics Teacher*. *45*(3)

Borba, M. and Confrey, J. (1996). A Student's Construction of Transformations of Functions in a Multiple Representational Environment. Educational Studies in Mathematics 31, pp. 319-337.

Boring, E.G. (1950). Great Men and Scientific Progress. *Proceedings of the American Philosophical Society*. *94*(4). pp. 339-351

Boyatzis, R. E. (1998) *The search for the codeable moment*. Transforming Qualitative Information. Thousand Oaks: Sage. Chapters 1 and 2

Bransford, J.D. & Schwartz, D. (1999). Rethinking transfer: A simple proposal with multiple implications. In A. Iran-Nejad & P. D. Pearson (Eds.), Review of Research in Education (Vol. 24 pp. 61-100). Washington, DC: American Educational Research Association.

Breidenbach et al. (1992). Development of the process conception of function. *Educational Studies in Mathematics*. *23*(3), pp. 247-285

Brenner et al. (1997). Learning by Understanding: The Role of Multiple Representations in Learning Algebra. *American Educational Research Journal*. *34*(4), pp. 663-689

Breslich, E. R. (1928). Developing Functional Thinking in Secondary School Mathematics. In *National Council of Teachers of Mathematics*, *The Third Yearbook* (pp 42-56). New York: Little & Ives.

Burrill et al. (2002). *Handheld Graphing Technology in Secondary Mathematics*. Retrieved April 3rd, 2010 from http://education.ti.com/sites/UK/downloads/pdf/References/Done/Burrill,G.%20(2002).pdf

Campbell, J.I.D. (Ed.) (1992). *The nature and origins of mathematical skills*. Amsterdam: North-Holland.

Carpenter, T. P. et al. (1981). Results from the Second Mathematics Assessment of the National Assessment of Educational Progress, NCTM, Reston, Virginia.

Carreira et al. (2002). *Mathematical Thinking: Studying the Notion of Transfer*. Proceedings of the International Group for the Psychology of Mathematics Education.

Carraher et al. (1985). Mathematics in the streets and in schools. *British Journal of Developmental Psychology*. *3*(1), pp. 21-29

Charles, R. I., & Silver, E. A. (1988). *The teaching and assessing of mathematical problem solving*. National Council of Teachers of Mathematics, Reston, VA.

Chi, M. Feltovich, P.J., Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science*, *5*. pp. 121-152

Clements, D. (1999). The future of educational computing research: The case of computer programming. *Information Technology in Childhood Education,* pp. 147-179

Clements, D. & Meredith, J. (1993). Research on Logo: Effects and efficacy. *Journal of Computing in Childhood Education, 4*, pp. 263-290.

Common Core Standards for Mathematics, retrieved January 29th, 2015 from http://www.corestandards.org/wp-content/uploads/Math_Standards.pdf

Confrey, J. (1991). The Use of Contextual Problems and Multi-Representational Software to Teach the Concept of Functions. Retrieved from ERIC database. (ED348229).

Confrey, J. & Costa, S. (1996). A critique of the selection of "mathematical objects" as a central metaphor for advanced mathematical thinking. *International Journal of Computers for Mathematical Learning*. *1*, pp. 139-168

Cooper, S., Dann, W. & Pausch, R. (2000) Alice: a 3-D tool for introductory programming concepts. *Journal of Computing Sciences in Colleges, 15*(5) pp. 108-117

Curcio, F. (1987). Comprehension of Mathematical Relationships Expressed in Graphs. *Journal for Research in Mathematics Education*. *18*(5), pp. 383-393

Cunningham, R. (2005). Algebra Teachers' Utilization of Problems Requiring Transfer between Algebraic, Numeric, and Graphic Representations, *School Science and Mathematics*. *31*(4), pp. 500-507

Detail on mathematics graduation requirements from public high schools, by state [pdf]. (2013). Retrieved from http://changetheequation.org/sites/default/files/MathGraduationRequirements_table.pdf

DiSessa, A. A. (1997). Twenty reasons why your should use Boxer (instead of Logo). In M. Turcsányi-Szabó (Ed.), *Learning & Exploring with Logo*. Proceedings of the Sixth European Logo Conference. Budapest Hungary, pp. 7-27.

DiSessa, A. A. (2004). Metarepresentation: Native Competence and Targets for Instruction. *Cognition and Instruction*, *22*(3), pp. 293–331

Dorsey, M.F. & Hopkins, L.T. (1930) The influence of attitude upon transfer. Journal of Educational Psychology, Vol 21(6), Sep 1930, pp. 410-417.

Dreyfus, T. (1999). Why Johnny Can't Prove (with Apologies to Morris Kline). *Educational Studies in Mathematics, 38* (1), pp. 85-109

Dreyfus, T. & Eisenberg, T. (1982). Intuitive Functional Concepts: A Baseline Study on Intuitions. *Journal for Research in Mathematics Education. 13*(5), pp. 360-380

Drijvers, P. (2000). Students encountering obstacles using a CAS. *International Journal of Computers for Mathematical Learning. 5*, pp. 189-209

Dweck, C. S. (2007). Mindset: The new psychology of success. New York: Ballantine Books

Dubinsky, E. (1995). ISETL: A programming language for learning mathematics. *Communications on Pure and Applied Mathematics, 48* (9), pp. 1027-1051

Edwards, J. & Jones, K. (2006) Linking geometry and algebra with GeoGebra, *Mathematics Teaching*, 194, pp. 28-30.

Edwards, L. (1998) Embodying mathematics and science: Microworlds as representations. *The Journal of Mathematical Behavior, 17*(1), pp. 53-78

Eisenberg, T. &Dreyfus, T. (1994). On understanding how students learn to visualize function transformations. Research on Collegiate Mathematics, *4*, pp. 45-68

Ellington, A. (2003) A meta-analysis of the effects of calculators on students' achievement and attitude levels in precollege mathematics classes. *Journal for Research in Mathematics Education. 34*(5). pp. 433-463

Else-Quest, N.M, Hyde, J.S., & Linn, M.C. (2010). Cross-National Patterns of Gender Differences in Mathematics: A Meta-Analysis. *Psychological Bulletin, 136*(1). pp. 103-127.

Felleisen, M., Findler, R. et al. (2001) How to Design Programs

Felleisen, M., Findler, R. et al. (2004) The Structure and Interpretation of the Computer Science Curriculum. *Journal of Functional Programming*, *14*(4), pp. 365-378

Feurzeig, W., Papert, S. et al. (1970). Programming-languages as a conceptual framework for teaching mathematics. ACM SIGCUE Outlook

Fey, J.T. (1984). Computing and mathematics : The impact on secondary school curricula. Reston, VA: National Council of Teachers of Mathematics.

Fischer, K. W. (1980). A theory of cognitive development: The control and construction of hierarchies of skills. *Psychological Review, 87*, pp. 477-531.

Fischer, K. W. & Granott, N. (1995). Beyond One-Dimensional Change: Parallel, Concurrent, Socially Distributed Processes in Learning and Development. *Human Development – Basel*. *38*(6). pp. 302-314

Fong, A. B., Jaquet, K., & Finkelstein, N. (2014). Who Repeats Algebra I, and How Does Initial Performance Relate to Improvement When the Course Is Repeated? REL 2015-059. *Regional Educational Laboratory West*.

Fuson, K.C. & Bransford, J.D. (2005). Mathematical understanding: An introduction. In S.M. Donovan & J. D. Bransford (Eds). How students learn: History, mathematics, and science in the classroom. Washington, DC: National Academy Press.

Gagné, R. M. (1962) The acquisition of knowledge. *Psychological Review, 69*(4), 355-365.

Gagné, R. M. (1968) *Learning Hierarchies*. Educational Psychologist. *6*(1). pp. 1-9

Gagné, R.M. & Glaser, R. (1987) Foundations in Learning Research. In R.M. Gagné (Ed). Instructional Technologies: Foundations. Hillsdale, N.J.: Laurence Erlebaum Associates, Inc.

Harel, G., & Dubinsky, E.. (1992). The concept of function: Aspects of epistemology and pedagogy. Washington, DC: Mathematical Association of America.

Harel, I., & Papert, S. (1990). Software design as a learning environment. Interactive Learning Environments, 1(1), 1–32.

Hill, H. C., Ball, D. L., & Schilling, S. G. (2008). Unpacking Pedagogical Content Knowledge: Conceptualizing and Measuring Teachers' Topic-Specific Knowledge of Students. *Journal for Research in Mathematics*, *39*(4), pp. 372–400.

Hill, H. C., Rowan, B., & Ball, D. L. (2005). Effects of Teachers' Mathematical Knowledge for Teaching on Student Achievement. *American Educational Research Journal*, *42*(2), 371–406.

Hegarty, M., Carpenter, P. A., & Just, M. A. (1991). Diagrams in the comprehension of scientific text. In R. Barr, M. L. Kamil, P. B. Mosenthal, & P. D. Pearson (Eds.), Handbook of reading research (Vol. 2, pp. 641–668). New York: Longman.

Hegedus, S. J., & Kaput, J. (2003). The Effect of a SimCalc Connected Classroom on Students' Algebraic Thinking. *International Group for the Psychology of Mathematics Education*, *3*, 47–54.

Hegedus, S. & Kaput, J. (2004). An introduction to the profound potential of connected algebra activities: issues of representation, engagement, and pedagogy. *Proceedings of the 28th Conference of the International Group for the Psychology of Mathematics Education* (vol 3, pp. 129–136)

Helfand, Duke. (2013, August 26). Formula for Failure in L.A. Schools. Los Angeles Times. Retrieved November 7, 2007, from http://www.latimes.com/news/local/la-me-dropout30jan30,0,1678653.story

Herscovics, N. & Linchevski, L. (1994). A cognitive gap between arithmetic and algebra. *Educational Studies in Mathematics*. *27*(1), pp. 59-78

Hoyles, C., & Noss, R. (2003). What can digital technologies take from and bring to research in mathematics education. *Second international handbook of mathematics education*.

Ioannou, I., & Angeli, C. (2013). Teaching computer science in secondary education: a technological pedagogical content knowledge perspective. the 8th Workshop in Primary and Secondary Computing Education (pp. 1–7). New York, New York, USA: ACM. doi:10.1145/2532748.2532755

Judd, C. H. (1908). The relation of special training to general intelligence. Educational Review, 36, pp. 28–42

Kafai, Y. B., Franke, M. L., Ching, C. C., & Shih, J. C. (1998). Game Design as an Interactive Learning Environment for Fostering Students“ and Teachers” Mathematical Inquiry. *International Journal of Computers for Mathematical Learning*, *3*(2), 149–184. doi:10.1023/A:1009777905226

Kaput, J. (1985). Representation and Problem Solving: Methodological Issues. In E. A. Silver (Ed.) Teaching and Learning Mathematical Problem Solving: Multiple Research Perspectives (pp. 381-398). Hillsdale, NJ: Erlbaum.

Kaput, J. (1986). Representation and Mathematics. In C. Janvier (Ed) *Problems of representation in mathematics learning and problem-solving*. Hillsdale, NJ: Erlbaum.

Kaput, J.J. (1989). Linking representations in the symbol systems of algebra. In S. Wagner & C. Kieran (Eds.), *Research issues in the learning and teaching of algebra* (pp. 167-194). Reston,VA: National Council of Teachers of Mathematics. Hillsdale, NJ: Erlbaum.

Kaput, J. (1991). Notations and Representations as Mediators of Constructive Processes. In Glasersfeld, E. (Ed.) *Radical Constructivism in Mathematics Education* (pp. 53-73).

Kahn, K. (1996) Toontalk - an animated programming environment for children. *Journal of Visual Languages and Computing, 7*(2)**,** 197—217

Kieran, C. (1990). Cognitive processes involved in learning school algebra. In P. Nesher & J. Kilpatrick (Eds.), *Mathematics and cognition: A research synthesis by the International Group for the Psychology of Mathematics Education* (pp. 96-112). Cambridge England: Cambridge University Press.

Kieran, C., & Chalouh, L. (1993). Pre-algebra: The transition from arithmetic to algebra. In D. T. Owens (Ed.), *Research ideas for the classroom: Middle grades mathematics* (pp. 179-198). New York: Macmillan.

Kieran et al. (1996). Introducing Algebra by means of a Technology-Supported, Functional Approach. In Bedharz, N., Kieran, C. et al. (Eds.), *Approaches to Algebra: Perspectives for Research and Teaching*, Chapter 19

Kieran, C. & Sfard, A. (1999) Seeing through Symbols: The Case of Equivalent Expressions. *Focus on Learning Problems in Mathematics*. *1*(1). pp. 1-17.

Kirshner, D. (1989). The Visual Syntax of Algebra. *Journal for Research in Mathematics Education*. *20*(3). pp. 274-287

Kirshner, D. (2004). Visual Salience of Algebraic Transformations. *Journal for Research in Mathematics Education*. *35*(4)

Knuth, E. (2000). Understanding Connections between Equations and Graphs. *The Mathematics Teacher, 93* (1), 48-53.

Knuth, E., Alibali, M., et al. (2005). Middle school students' understanding of core algebraic concepts: Equivalence variable. *ZDM.Zentralblatt Für Didaktik Der Mathematik.Articles 37*(1). pp. 68-76.

Konidari, E. & Louridas, P. (2010) When Students are Not Programmers. *ACM Inroads*, *1*(1), pp. 55-60.

Kurland, D. & Pea, R. (1983). Children's mental models of recursive Logo programs. Proceedings of the Fifth Annual Meeting of the Cognitive Science Society.

Leron, U. & Dubinksy, E. (1995). An Abstract Algebra Story. *The American Mathematical Monthly*, *103*(2). pp. 227-242.

Lobato, J. (2003). How design experiments can inform a rethinking of transfer and vice versa. Educational Researcher, 32(1), 17-20.

Loveless, T. (2008). The misplaced math student: Lost in eighth-grade algebra. Washington, DC: Brown Center on Education Policy, Brookings Institution.

Marzano, R. J. (2008). *A theory-based meta-analysis of research on instruction*. Aurora, CO: Mid-continent Research for Education and Learning

Matz, M. (1982). Towards a process model for high school algebra errors. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. pp. 25-50. London: Academic.

Mayer, R.E. (2005). *The Cambridge handbook of multimedia learning*. New York: Cambridge University Press.

Mayer et al. (1995). A Comparison of How Textbooks Teach Mathematical Problem Solving in Japan and the United States. *American Educational Research Journal*. *32*(2), 443-460

Mevarech, Z. & Kramarsky, B. (1997) From verbal descriptions to graphic representations: Stability and Change in Students' Alternative Representations. *Educational Studies in Mathematics 32*(3), 229-263

Milner, S. (1973). *The Effects of Computer Programming on Performance in Mathematics*. Paper presented at the annual meeting of the American Educational Research Association, New Orleans, Louisiana.

Mishra, P., & Koehler, M. (2014). *Handbook of Technological Pedagogical Content Knowledge (TPCK) for Educators*. Published by The AACTE Committee on Innovation and Technology, Herring, M., Routledge.

Newell, A. & Simon, H. (1972) *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, NJ.

National Council of Teachers of Mathematics (1989). *Curriculum and evaluation standards for school mathematics*. Reston, VA

National Council of Teachers of Mathematics: (2000). *Principles and standards for school mathematics*. Reston, VA: National Council of Teachers of Mathematics.

National Science Board (2012). Science and technology: Public attitudes and understanding. *Science and Engineering Indicators 2012* (National Science Foundation, Arlington, VA).

Noss, R. (1986). Constructing a conceptual framework for elementary algebra through Logo programming. *Educational Studies in Mathematics*, *7*, 335-357

O'Callaghan, B. (1998) Computer-intensive algebra and students' conceptual knowledge of functions. *Journal for Research in Mathematics Education*. *29*(1), 21-40

Papert, S. (1980) “Mindstorms: Children, Computers, and Powerful Ideas”. Basic Books, New York.

Papert, S. (1972) Teaching Children to be Mathematicians Versus Teaching About Mathematics. *International Journal of Mathematical Education in Science and Technology*. *3*(3), 249 – 262

Pea, R. & Kurland, D. (1984) On the cognitive effects of learning computer programming. *New Ideas in Psychology, 2*, 137-168

Pea, R. (1987) Cognitive Technologies for Mathematics Education. In Schoenfeld, A. H. (Ed.) *Cognitive Science and Mathematics Education,* Hillsdale, NJ.: Erlbaum.

Perkins, D. (2009). Making Learning Whole. Jossey-Bass, San Francisco

Piaget, J. Grize, J.B., Szeminska, A., et al. (1977) Epistemology and Psychology of Functions. Reidel, Boston.

Ponce, G. A. (2007). Critical Juncture Ahead! Proceed with Caution to Introduce the Concept of Function. *Mathematics Teacher*. *101*(2), 136-144

Resnick, B. et al. (1981). *The Psychology of Mathematics for Instruction*. L. Erlbaum Associates

Resnick, M. et al. (2009) Scratch: Programming for All. *Communications of the ACM*. *52*(11). 80-87

Roschelle, J., Tater, D., Shechtman, N. (2007). Scaling up SimCal: Can a technology-enhanced curriculum improve student learning of important mathematics? Technical Report, SRI International: math.sri.com

Reed, S.K., Ernst, G.W. & Banerji, R. (1974). *The role of analogy in transfer between similar problem states*. Cognitive Psychology. *6*(3). pp. 436-450

Rose, H. & Betts, J. (2004). *The Effect of High School Courses on Earnings*. The Review of Economics and Statistics

Rosenbloom, P.S., Laird, J.E., et al. (1985) R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. *PAMI-7*(5). pp. 561-569

Schaaf, W. (1930). Mathematics and World History. *Mathematics Teacher*, *23*, 496-503.

Schanzer, E., & Star, J.R. (2010). Teaching on the wall of Plato's cave: How representational affinity impacts understanding of algebraic functions. P. Brosnan, D.B. Erchick, & L. Flevares, (Eds.). (2010). Proceedings of the 32nd annual meeting of the North American Chapter of the International Group for the Psychology of Mathematics Education (p. 284). Columbus, OH: The Ohio State University.

Shulman, L. S. (1986). Those Who Understand: Knowledge Growth in Teaching. *Educational Researcher*, *15*(2), 4–14.

Schwarz, B. & Dreyfus, T. (1995) New actions upon old objects: A new ontological perspective on functions. *Educational Studies in Mathematics*. *29*(3), 259-291

Schwartz, J. & Yerushalmy, M. (1985) *The Geometric Supposer*. Pleasantville, NY

Sfard, A. (2000) Steering (Dis)Course between Metaphors and Rigor: Using Focal Analysis to Investigate an Emergence of Mathematical Objects. *Journal for Research in Mathematics Education 3*(21). 296-327

Sfard, A. & Linchevski, L. (1994) The gains and the pitfalls of reification—The case of algebra. *Educational Studies in Mathematics 26*(2-3), 191-228

Silver, E.A. (1979). Student Perceptions of Relatedness among Mathematical Verbal Problems**.** *Journal for Research in Mathematics Education*. *3*(10). pp. 195-210

Singley, M.K. & Anderson, J.R. (1989) The Transfer of Cognitive Skill. Cambridge, MA. Harvard University Press

Slavit, D. (1997). An Alternate Route to the Reification of Function. *Educational Studies in Mathematics*. *33*(3), 259-281

Star, J. R., & Rittle-Johnson, B. (2009). Making Algebra Work: Instructional Strategies that Deepen Student Understanding, within and between Algebraic Representations. *ERS Spectrum*, *27*(2), 11–18.

Stephenson, C., Barr, V. (2011). "Defining Computational Thinking for K-12". *CSTA Voice*. *7*(2). pp. 3–4

Sutherland, R. (1989) Providing a Computer Based Framework for Algebraic Thinking. *Educational Studies in Mathematics, 20*, 317-344.

Swafford, J. O. & Brown, C. A. (1989) Variables and relations. In M. Montgomery Lindquist (Ed.), *Results from the Fourth Mathematics Assessment of the National Assessment of Educational Progress* (pp. 55-53), NCTM: Reston, Virginia.

Thompson, A. G., & Thompson, P. W. (1996). Talking about Rates Conceptually, Part II: Mathematical Knowledge for Teaching. *Journal for Research in Mathematics Education*, *27*(1), 2.

Thorndike, E. L., Woodworth, R. S. (1901) The influence of improvement in one mental function upon the efficiency of other functions. Psychological Review 8. pp. 247-261

Usiskin et al. (1988). Conceptions of school algebra and uses of variables. *The Ideas of Algebra, K-12* (pp. 9-19).National Council of Teachers of Mathematics, Reston, VA.

U.S. Department of Education, National Center for Education Statistics, Education Longitudinal Study of 2002 (ELS:2002), "Base Year, 2002" and "First Follow-up, 2004."

U.S. Department of Education, National Center for Education Statistics, High School and Beyond Longitudinal Study of 1980 Sophomores (HS&B-So:80/82), "High School Transcript Study"; and 1987, 1990, 1994, 1998, 2000, and 2005 High School Transcript Study (HSTS). (This table was prepared January 2007.)

Van Eck, R. (2001). *Promoting Transfer of Mathematics Skills through the Use of a Computer-Based Instructional Game and Advisement*. Retrieved November 2nd, 2007 from http://www.mtsu.edu/~itconf/proceed01/13.pdf

Van Lehn, K. (1990). Mind Bugs: The Origins of Procedural Misconceptions. Cambridge, MA. MIT Press.

Voss, J. (1987). Learning and transfer in subject-matter learning: A problem-solving model. *International Journal of Educational Research, 11*(6), pp. 607-622

Wagner, S., & Kieran, C. (Eds.). (1989). *Research issues in the learning and teaching of algebra*. National Council of Teachers of Mathematics, Reston, VA.

Wertheimer, R. (1990). The Geometry Proof Tutor: An "Intelligent" Computer-based Tutor in the Classroom. *The Mathematics Teacher*. pp. 308-317

Wilson et al. (1994). One Preservice Secondary Teacher's Understanding of Function: The Impact of a Course Integrating Mathematical Content and Pedagogy. *Journal for Research in Mathematics Education*. *24*(4), pp. 346-370

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*. *49*(3). pp. 33–35.

Wright, G., Lee, R., Rich, P. & Leatham, K. (2011). *An Analysis of the Influenceon Students' Mathematics Skills Participating in the Bootstrap Programming Course*. In Proceedings of World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education 2011 (pp. 986-991)

Yan, Z., Fischer, K.W. (2002) Always Under Construction: Dynamic Variations in Adult Cognitive Development. *Human Development – Basel*. *45*(3). *pp. 141-160*

Yoo, D., Schanzer, E., Krishnamurthi, S. et al. (2011). *WeScheme: the browser is your programming environment*. In Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, pp. 163–167. ACM.

Yerushalmy, M. (1991) Student perceptions of aspects of algebraic function using multiple representation software. *Journal of Computer Assisted Learning*, 7, *42-57*

Zheng, T. (1998). Impacts of using calculators in learning mathematics. *The 3rd Asian Technology Conference on Mathematics (ATCM'98)*. Retrieved May 16th, 2010 from: http://www.any2any.org/EP/1998/ATCMP015/paper.pdf

Vita

Emmanuel Tanenbaum Schanzer

1998-2002	Cornell University Ithaca, NY	B.A. May, 2002
2002	Program Manager, Microsoft Corporation Seattle, WA	
2003	Teacher, MATCH School Boston, MA	
2004-2006	Teaching Fellow, Citizen Schools Boston, MA	
2004-2006	Lesley University, Boston, MA	M. Ed. May 2006
2007-2015	Doctor of Education Candidate Graduate School of Education Harvard University	