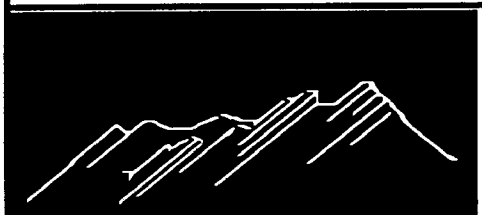
INSTITUTE OF COGNITIVE SCIENCE



Technical Report

University of Colorado, Boulder

Challenging Students' Misconceptions of Higher Order Mathematics: Visualizing Functions as Data Objects

Julie DiBiase

Institute of Cognitive Science
Department of Computer Science
University of Colorado
Boulder, CO 80309

Technical Report 95-07

Challenging Students' Misconceptions of Higher Order Mathematics: Visualizing Functions as Data Objects

Julie DiBiase
Department of Computer Science
University of Colorado, Boulder

In this article, I discuss changes in students' concept of function over the course of their participation in a curriculum based on *multi-modal imagery building*. Students between the ages of ten and fifteen participated in individual tutoring sessions focusing on a mathematically advanced application of function which is typically not addressed before the college level. This work examines the effects of the imaging curriculum on students' concept aquisition. Results indicated that younger students (1) exhibited stereotypical novice errors, yet (2) were able to acquire and apply an expert concept of function. Pedagogical strategies for applying imaging tools to abstract concepts are discussed.

> "In mathematics... we find two tendencies present. On the one hand, the tendency toward *abstraction* seeks to crystallize the *logical* relations inherent in the maze of material that is being studied... On the other hand, the tendency toward *intuitive understanding* fosters a more immediate grasp of the objects one studies, a live *rapport* with them, so to speak, which stresses the concrete meaning of their relations..." (Hilbert 1932)

Historically, functional programming and higher mathematics have been characterized by their almost relentlessly textual and axiomatic style of teaching (Bradley, 1995; Cunniff, Taylor, & Black, 1989; Davis, 1951; Ralston & Young, 1983); the educational tradition for these topics is largely devoid of the rich "manipulatives" and visual images that characterize much of the most creative work in basic mathematics education. This work focuses on developing students' intuitions about higher-order abstractions through the use of imaging tools.

Informally, the main idea examined here is that *functions*[1] *are manipulable as units of data.* Functional data objects, by definition, share the same **four properties** as other "first-class" data objects in a programming language (e.g. numbers): they can be (1) named; (2) passed as arguments to other functions; (3) returned as the results of function calls; (4) used to form complex data structures[2] (Stoy 1977).

The concept of functional data, central to the whole paradigm of functional programming, can be traced back far into the foundations of computer science: it is at the heart of Turing's universal machine concept (Turing, 1950). Although a means for the robust and elegant expression of ideas (Babbage 1842; Gödel 1931; Turing 1937; Von-Neumann 1945; Aleksandrov 1963) it has simultaneously been a source of confusion (Babbage, 1842; DiBiase, 1995b). Computer programming courses have been sufficiently intimidated by the concept, ignoring it or relegating its proper treatment to advanced courses (Wilensky, 1986; Winston & Horn, 1989). In the first specific

[1] For the purposes of this work, *functions* are defined in a decidedly "procedural" manner. Although strictly speaking the terms "function" and "procedure" have a subtle semantic distinction, they are herein used interchangeably and indistinguishably. Other alternative representations of functions (e.g. graphs, sets of pairs) are less viable in the current context of this work.

[2] Property 4 is not examined by this work.

investigation of the higher-order function concept within the domain of programming, Eisenberg, Resnick and Turbak (1987) studied freshmen enrolled in an introductory programming course at MIT in which functional data served as a foundation for the course. Despite the course's philosophy, over 50% of MIT freshman were nevertheless unable to successfully apply this concept. Figure 1 shows a prototypical example of students' misconception.

This work seeks to unveil the causes of the difficulties experienced with overwhelming regularity by students of higher-order function. A curriculum is developed which directly seeks to curtail the standard taxonomy of misconceptions about functional data. The behavior of students involved in the preliminary application of this type of imaging curriculum provides an optimistic view of its utility for abstract concept acquisition.

```
a. >>> (define (apply-to-5 f) (f 5))
b. >>> (define (create-subtracter n) (lambda (x) (- x n))
c. >>> (apply-to-5 create-subtracter)
```

Figure 1 In several separate surveys of functional programmers, students incorrectly reported that the last expression in above sequence resulted in an error (DiBiase, 1995b; Eisenberg, Resnick, & Turbak, 1987). Expression **a** defines a function (cf. Stoy, Property 1), `apply-to-5`, which takes a unary function, `f`, as its argument (Property 2); `apply-to-5` then applies the specified unary function to the number 5 and returns a result. The second expression takes a numerical argument, `n`, and creates an unnamed function (Property 3), the result of the entire expression in **b**, which decrements a numerical argument, `x`, by that number (n). The third expression integrates all three properties of higher-order function: to solve it correctly, students must understand that a function can behave as the input and output of other functions. In calculus pedagogy, derivatives provide another prototypical example of this concept and its phenomenological misconceptions.

THE DIFFICULTY WITH FUNCTION

Functions, in both a general and mathematical sense, are typically viewed as active entities: to function is to "be in action" (Oxford English Dictionary, 1958). Functional objects necessitate a dualistic conceptualization that encompasses both their active and passive nature. Other examples of this type of dual categorization are hard to find, making it difficult to intuit: fundamentally, even natural language makes a clear distinction between nouns ("person, place or thing") and verbs ("action words") (Allen, 1987). Chi points out that conceptual category change is most difficult when the naive categorizations are deeply rooted through these same types of persistent, consistent and recapitulated support systems (Chi, Slotta, & Leeuw, 1994). Instinctual problems are perpetuated by current day computer programming paradigms which neglect constructs for supporting functional objects. Even some functional languages support a naive concept of function. For example, in Common LISP, the same symbol can be attached both to a function and a value, necessitating the use of the LISP `funcall` primitive. As a result, procedures can not be directly abstracted in the same way that numbers can indicating, in some sense, a different status (Winston & Horn, 1989).

A recent anthology published by the Mathematical Association of America dedicated itself entirely to examining the well known but sparsely documented problems with the concept of function (Harel & Dubinsky, 1992). Dubinsky and colleagues have postulated an epistemology of functions (Breidenbach, Dubinsky, Hawks, & Nichols, 1992) based on a three phase model:

1. *Action*: the ability to plug numbers into an algebraic expression and calculate.
2. *Process*: dynamic transformation of quantities according to some repeatable means.
3. *Object*: the ability to perform actions on and transform the function itself.

Several studies have targeted computer programming models as aids in the development from *action to process* concept of function (Ayer, Davis, Dubinsky, & Lewin, 1993; Breidenbach, et al., 1992; Cuoco, 1993b; Cuoco, 1995). Little work has been done, however, to trace the development from *process to object* concept of function. In one of the only such scenarios, a 14 year old child participated in a twelve week study in which researchers attempted to use a computer environment (with which the child was already proficient) to examine the student's development from process to object concept of function (Kieran, Garaicon, Lee, & Boileau, 1993); the study reports that the child did not acquire an object concept of function.

Based on Chi's theories of conceptual change (Chi, Slotta, & Leeuw, 1994), students' misunderstandings about the object nature of function are not surprising. According to the "incompatibility hypothesis" certain science concepts are troubling because of a difference between the categorical representation that students bring to an instructional context and the ontological category to which the science concept truly belongs (Matter, Process or Mental State). The more difficult entities to learn about are those that simultaneously embody more than one category and hence require conceptual alternation, e.g. matter and process (or, if you will, object and function).

IMAGERY AND ABSTRACTION

The key to unlocking problems with higher-order function lies in mastery of abstraction. Sfard (1992) puts it too well to rephrase in her paper "Operational Origins of Mathematical Objects and the Quandary of Reification -- The Case of Function" :

> *The Ideal mathematician*, according to Davis and Hersh (1983, p. 35), *studies objects whose existence is unsuspected by all except a handful of his fellows.* Even the strangest abstract entities, while scrutinized and manipulated, seem to the mathematician as unquestionably real as the pen with which he or she writes papers...Shortly after being introduced to [the notion of function, the student] is expected to analyze and manipulate the new entity with a confidence which can only be achieved by those who can treat it as if it were a real thing. Many of our students, however, seem to be lacking this ability. Recent studies on learning mathematics abound in findings which can serve as evidence. Awareness of the long and painful process preceding the birth of a mathematical object may be the key to understanding some of the difficulties experienced by so many learners.

The difference, as noted by Sfard above, between an expert and a novice mathematician lies in the level of comfort with abstract objects. Indeed, the key to expertise rests in the ability to mentally manipulate abstract objects *as if they were real objects.* The imaging curriculum stresses the relationship between this ability and one's expertise in mental imagery. The task for educators, then, is to build more concrete, accessible representations which can act as stepping stones, carrying the student from novice to expert abstracter.

This type of epistemological analysis is hardly new, dating back to Piaget's theory of *reflective abstraction* (1966)[3] and earlier to Hadamard's reflections specifically about introspection and invention in the mathematical field (Hadamard, 1944). These types of

[3]To be precise, the very earliest roots of imagery in pedagogy are traceable to the writings of Plato (Waterfield, 1974).

analysis likewise stressed the use of concrete representations - images, drawings or linguistic artifacts - to catalyze the generation of logical and intuitive imagery. One of the goals of this work is to enhance imagery ability and to justify the feasibility of this approach; thus we turn towards a more contemporary summary of some of the important empirical results in the study of mental imagery.

Imagery researcher David Marks claims that while the *ability* to generate and employ mental imagery varies across people, the *potential* to do so is universal. Results of a study on picture-memory task retrieval indicated that "poor" visualizers produced 36% more errors than good visualizers. Ability to visualize, Marks asserts, depends on situational variables which can be manipulated to enhance imagery ability (Marks, 1972). This is supported by (Marschark, 1988) who also claims that mediation (metacognitive instruction) is useful in fostering children's imaging ability. According to Marks, imagery is useful as an associative retrieval aid between different but related stimuli (Marks, 1990).

In studies linking imagery and problem solving, Kaufmann found that imagery is most important in the initial phases of the task; with increasing familiarity, a purely linguistic representation was favorable. Novelty, complexity, and ambiguity all seem to place additional cognitive demands on a task (Kaufmann, 1988b). In a second study, Kaufmann linked imagery specifically to the discovery stage of problem solving: utility of imagery increased systematically as the level of programming (familiarity) in the task to be solved decreased (Kaufmann, 1990; Kaufmann, 1988a). Engelkamp found that the effect of added exposure to the recall stimulus (planning, executing, visualizing, *and* verbalizing as opposed to planning, executing, visualizing *or* verbalizing) is cumulatively positive (Engelkamp & Zimmer, 1990).

In a study of imagery and unexpected learning, subjects instructed to image and not to learn were unable to prevent learning (Sheehan, 1972). Research has conclusively correlated imaging ability to high performance on memory tasks involving concrete stimuli (Paivio, 1971). Paivio also discovered that "high" imagers reacted significantly more quickly than low imagers when the stimulus words were abstract. He concludes that abstract language does not evoke imagery as readily as concrete. In another study, results indicate that differences in recall between concrete and abstract stimuli are reduced or even eliminated when rich contextual factors operate (Paivio, 1988).

More recently, additional research on representation and action words has positively linked performance and motor imagery (imaging performance) to efficient verb encoding and comprehension (Engelkamp, 1988). This study found that verbs were learned much less easily than nouns under standard instruction (63 vs. 72 percent) but that this difference was eliminated when the learning included motor encoding (acting) (77 vs. 79 percent). They further discovered that elaboration (acting more, longer, or differently) had no effect on the initial success, indicating that motor encoding had a particularly strong initial effect which was not readily improved through further processing.

To summarize, the above research concludes that imaging is positively linked to learning. Although less inherently developed for abstract tasks, this is nevertheless the domain in which imagery is most strongly needed (Kaufmann, 1988a). Imagery is particularly useful in the initial (novel) stages of problem solving. Motor imagery has also been shown to be a particularly strong method of encoding.

It would be negligent to leave this topic without acknowledging that the debate over mental imagery is far from resolved. The raging debate (Tye, 1991), is focused mainly on the exact *nature* of mental imagery. The issue is representational: what is the *composition* of an image? Tye outlines a set of competing theories on the nature of imagery, including: (a) *the picture theory* -- mental images are pictures, similar to projections on a computer screen (Kosslyn, 1980); (b) *structural descriptionalism* -- mental images are complex linguistic representations (Pylyshyn, 1981); and (c) *array theory* -- mental images are interpreted, symbol filled arrays (Tye, 1991).

While the debate about representation rages on, the single denominator among theories seems to be the notion that, whatever its nature, imagery does exist as an introspective phenomenon. Work on imaging curricula does not rely on the resolution of the imagery debate: the empirical results that will be presented are sufficiently promising indicators of the utility of imagery as a tool for learning.

THE IMAGING CURRICULUM

The preceding discussion defines the core pedagogical problem for students of higher-order function: conceiving of functions as data objects in the face of their veritable abstractness. The *multi-modal imaging curriculum* is designed to present students with a variety of representations (cf. Englekamp; Gardner, 1993; Polya, 1945) -- from concrete to abstract -- which serve as aids for building a mental representation of functions as both passive and active entities. Basic math educators have for years applauded the use of "manipulatives" for reifying abstract concepts (Hughes, 1986; March, 1977; Montessori, 1956; Papert, 1980; Thyer & Maggs, 1971); this work extrapolates two-dimensionally by (a) extending these principles to abstract mathematics, and (b) relying on a library of tools for any particular concept. Cuoco (1995) describes a series of computational tools for representing functions, concluding that none of the computational media available expresses "every facet of the function concept". In a multi-modal learning environment any one tool *cannot* represent -- or at the very least *is not required* to represent -- the comprehensive embodiment of a concept.

For the concept of higher-order functions, students worked with three types of imaging tools, in increasing degrees of concreteness: (1) computational; (2) pictorial; (3) kinesthetic. Students were able to participate in the "creation" of artifacts with each type of tool (Harel & Papert, 1991). As their computational medium, students used SchemePaint, a graphics enhanced version of the Scheme programming language with direct manipulation extensions (Eisenberg, 1991; Eisenberg, Clinger, Harheimer, & Abelson, 1990). Students were first introduced to the behavior of the Scheme interpreter through a two dimensional visual representation, the Name-Object Table (Figure 2).

Shortly thereafter, students are presented with the third and most concrete representation to work with: these are small felt pillows with it's own textual information inside (miniature dry-erase boards inscribed with a copy of the 2-D representation). Students were able to choose which aids they preferred for expression and problem solving. The ultimate goal for the student always involved using the computer application to create graphics artifacts; research goals were to monitor how effectively students were able to apply an *object concept* of data.

METHOD

Participants

A total of 18 students[4] were recruited from several sources: five were children of colleagues; nine were high school students working on independent projects; three were middle school children working independent of school projects; one was a university undergraduate. All students attended the local Colorado public school system; only one was a minority student. Eight of the students had little or no experience working with computers; six were well versed at using applications, like games or word processors; four had some prior programming experience in BASIC, Pascal, or C. None of the students were previously proficient programmers. All students participated in the project of their own free will with no remuneration.

[4]Three of the case studies actually took place in a pre-design phase, without the aid of the imaging curriculum.

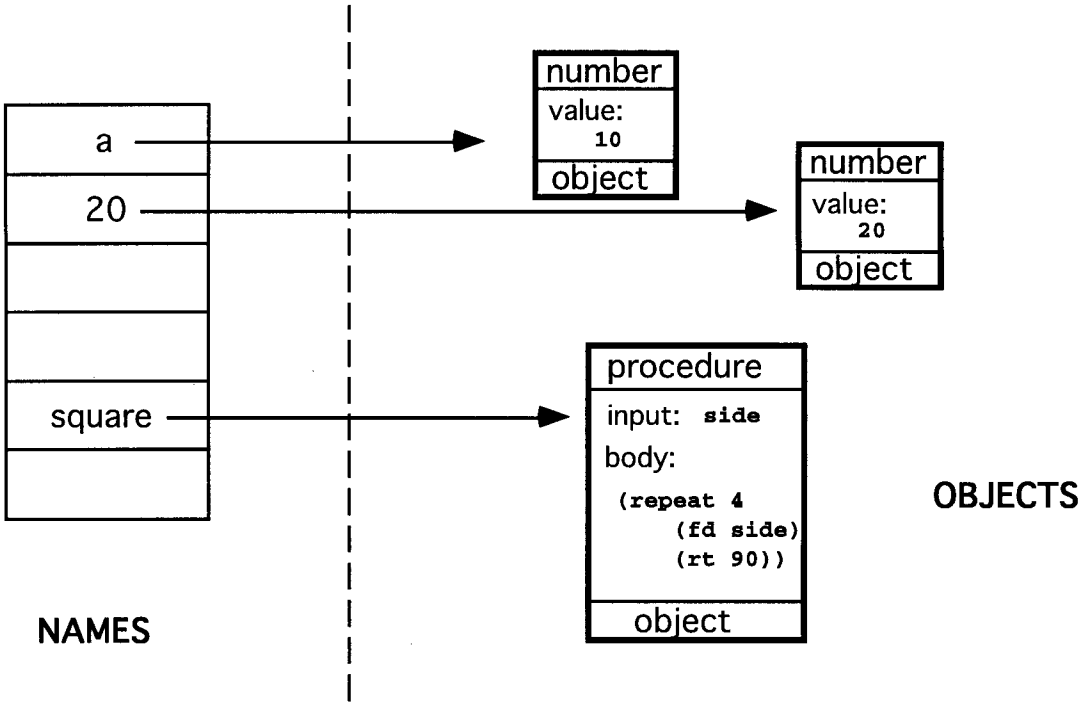


Figure 2 This pictorial representation of the Name-Object Table is displayed on a dry-erase board and available to students to modify as a tool for problem solving.

Setting

The setting for the curriculum was a small research office at the University of Colorado, Boulder containing: (a) a computer running SchemePaint; (b) a large dry-erase board; (c) concrete manipulative objects.

Students worked individually with an instructor for a total of 10-20 hours. The duration and pace of a session varied with the individual -- sessions ranged from 30-120 minutes in length.

Data Collection

Case studies were performed in four iterations over a two and a half year period. This work reports on the performance of the six students who participated in the final iteration of the curriculum. Data was collected from the students in three ways:

1. Written Questionnaires. Students were periodically required to complete questionnaires that tested their acquisition of certain concepts. Appendix A shows an actual questionnaire that was completed by a student.
2. Session Transcripts. SchemePaint saves a computerized transcript of a user's interaction with the interpreter; written notes of non-computational interactions were integrated into transcripts at the end of each session.
3. Audio Taped Interviews. During the last third of the study, students were directly asked to reflect on their impressions of the curriculum, what they had learned, and how useful the imaging tools were.

RESULTS

Case Study Sample

Gretta was a 14 year old high school student with little interest or background in mathematics. She chose to do the SchemePaint project to fulfill a workshop requirement at school and because her father wanted her to learn more about computers. She was quiet and introverted. We met for eight weeks in two half-hour sessions per week. Her progress in the curriculum typified the behavior of a successful student.

To begin with, all students were asked to think about their impressions of objects in the world: as an example of a thing which was both active and passive Gretta cited "a blender". Immediately following that, students were given an introduction to objects in the SchemePaint environment. Figure 1 showed the two most common types of data objects in SchemePaint, *number* and *procedure*. Students are able to use the 2-D or 3-D representation of objects, combined with "rules of evaluation" to model the behavior of the Scheme interpreter (Figure 5 will describe an example of this procedure).

Armed with just a basic repertoire of SchemePaint graphics commands, Gretta was encouraged to explore her own interests. She spent several begining sessions working out a sequence of expressions to draw a circle. At first, she merely tinkered with the numbers in her expressions in order to customize the circle. With instruction, she was taught to define this series of commands into a unit, that is, a function; Gretta was subsequently introduced to arguments (inputs) which she then incorporated into her circle function in order to enable the the user to vary the size of a circle at run time. Shortly thereafter, *Gretta asked if it was "ok" to replace the* `right` *procedure with another input so that she could specify whether she wanted the circle to turn* `left` *or* `right` *on each run.* She used the modifed procedure to make "eyes" (Figure 3). The progression of Gretta's work is explained in detail in Figure 4.

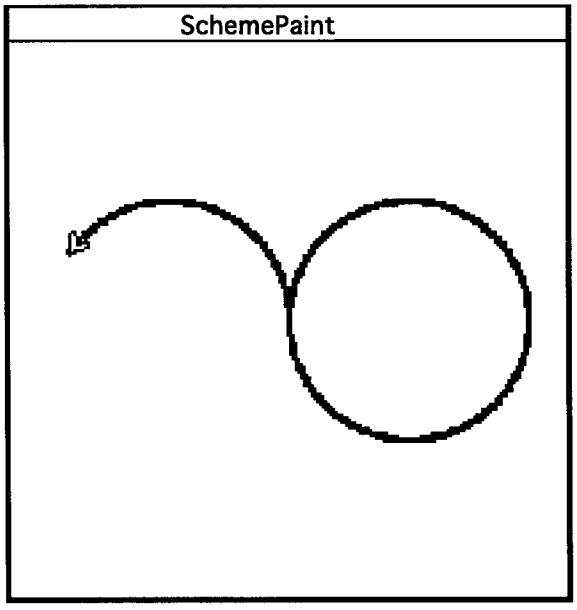


Figure 3: The "eyes" that result from the commands `(circle 2 rt)` and `(circle 2 lt)`.

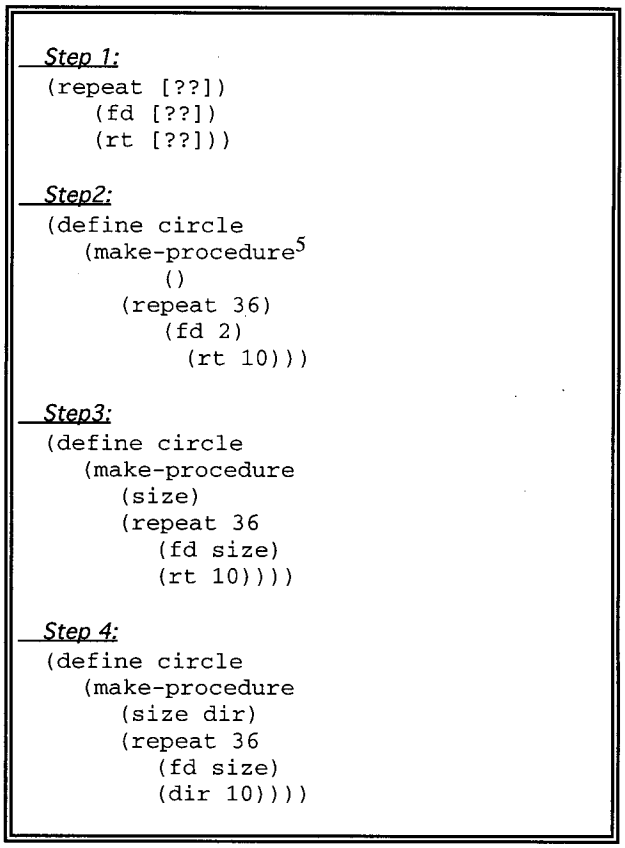


Figure 4: The four step process Gretta used to perfect her `circle` function. In step 1, she explores the circle's geometry, varying the grain at which it is drawn, by repeatedly typing in the same basic series of commands while varying different numerical values. In step 2, she learns to `define` a sequence of commands as a functional unit; step 3 extends this convience through a user-input parameter (the `size` of the circle). Once step 3 is complete, the user can draw circles of varying size with such simple commands as `(circle 3)`. By step 4, Gretta has generated the notion of functions as input to other functions on her own. The circles drawn in Figure 3 are the result of two commands which use the procedure defined in step 4: `(circle 2 right)` and `(circle 2 left)`.

In this exercise, Gretta generated -- without being prodded -- the concept that a function could behave as a data object, to be "passed" as an argument to another function (cf. Stoy, Property 2 of first class data objects). This result becomes even more astounding by comparison to other preliminary studies, where 57% of undergraduates studying Scheme in a university level programming languages course were unable to correctly answer a problem which simply applied this principle (DiBiase, 1995b).

Students typically have even more trouble with Stoy's Property 3 (functions can return other functions as results): in the same study cited above, failure rates of 64-79% were recorded on a range of related problems. More importantly, the types of errors observed were extremely uniform. One particularly troublesome concept was that of an anonymous procedure (the sort generated by a `lambda` expression -- or in

[5] Students used the macro `make-procedure` to replace the traditional lambda expression.. The syntax shown here was used to unify the definition of all objects in the language.

this case, `make-procedure`). Like most students, Gretta was not immune to such errors. In her very first exposure to Property 3, she was asked to write a procedure called `make-colored-hexagon-procedure` (`mchp`) which took a color object as input and generated a "hexagon making procedure". Her first attempt was just shy of correct: interestingly, she tried to unnecessarily define and assign a name to the generated procedure, indicating that she had a problem with its anonymity . As the next section will discuss, anonymous procedure objects are the most common difficulty for students of functional programming. To debug this and other problems, Gretta chose to use 3-D manipulatives. In fact, Gretta always wrote and verified her functions using this brand of tools -- before, during and after she entered a computational version.

Figure 5 describes (in an unjust fashion, unfortunately, in the absence of the physical manipulatives) the correct description of the execution of a call to `mchp`. Steps 1-4 on the left side are referred to as *Rules of Evaluation*. Steps 1-4 on the right are typically performed by students using concrete object manipulatives; in other words, the step "look-up the input" involves the student actually moving across the room to retrieve an object. Such step-by-step run-throughs with manipulatives allow students to reconcile the validity of an anonymous object.

At the end of nine weeks Gretta was interviewed about her opinions regarding the work we had done. Gretta's self-assessment was consistent with her observed working behavior. Four interesting excerpts are shown in Figure 6.

As a 14 year old novice programmer, Gretta was able to generate the idea of property 2 all on her own. She dabbled with property 3 and became proficient at applying this property with the aid of the physical manipulatives. Her combined experience from work, tests, and interviews supports a complex view of function, including the following premises: (1) procedures, like numbers can be variable; (2) non-computational represenatations can frame computational problems in a more accessible fashion; (3) concrete representations solidify and demystify abstract concepts.

Interviewer: Does it seem any different to use procedure objects as inputs than it does to use number objects as inputs?
Gretta: *Kind of the same but there is a bit of a difference. Both can be variable.*

Interviewer: Sometimes we represent objects as pictures on the board in the framework of the name/object table. Sometimes we represent them with the puffy objects. Do you think either of these representations has helped you to learn about objects in SchemePaint?
Gretta: *Yeah. It helps to show what's going on so that we don't always think of it as a big mass of numbers. It helps you understand what's going on so that you don't just think of the computer as this big massive thing that shoots something out at you.*

Interviewer: So, do the objects help make the abstract more understandable?
Gretta: *Yah. Humans make minds think they should see everything concretely so it helps when you can see it concretely.*

Interviewer: What do you think the objects are good at showing?
Gretta: *How it goes and gets them, has to look them up and put them together. It shows it in a more solid way.*

Figure 6 Excepts from an interview describe a solid object concept of data.

```
>> (define mchp
     (make-procedure
       (col)
       (make-procedure
         (size)
         (set-default-color col)
         (repeat 6
           (fd size)
           (rt 60))))
```

To Evaluate:

(mchp blue)

1. Look-up the input:

color object					
R:	0	G:	0	B:	65000

2. Look-up the procedure:

procedure object	
INPUT:	col
BODY:	(make-procedure (size) (set-default-color col) (repeat 6 (fd size) (rt 60))))

3. Associate the real input with its symbolic name from procedure in step 2:

procedure object	
INPUT:	~~col~~ **blue**
BODY:	(make-procedure (size) (set-default-colo**blue** (repeat 6 (fd size) (rt 60)))

4. Sequentially evaluate the instructions in the body of step 3 to produce the final result:

procedure object	
INPUT:	size
BODY:	(set-default-color blue) (repeat 6 (fd size) (rt 60)))

Figure 5 The visual execution of `(mchp blue)`. The correct definition of mchp is shown at the top of the figure. Students are asked to describe the evaluation of the expression: `(mchp blue)`. On the left hand side are four general rules for evaluating any function call. On the left hand side are visual results of these steps for the expression in question. Students may choose to work through problems with this visual representation or use physical three dimensional manipulatives which also contain a copy of this textual representation. In step 1, the student looks up all the input to the function being called; in this case, the color object corresponding to the name `blue` is retrieved. In step 2, the object corresponding to the function `mchp` is also retrieved. With these objects in hand, students can begin step 3: associating the actual input for this exercise (`blue`) with the formal parameter specified in the body of the function object being called (`color`). Now, in step 4, with a fully specified procedure object, students can begin sequentially executing the expressions in the procedure body. The result of `(mchp blue)`, then, is shown at the bottom of th right hand column: it is also a procedure object.

General Results

Table 1 summarizes the results from the final iteration of the curriculum (for a complete report on all 18 subjects see (DiBiase, 1995a)).

TABLE 1
Student Achievement Summary.

ALIAS	AGE	LEVEL	TIME	(apply-to-5 create-subtracter)
nate	15	III+	21	yes
aaron	15	III+	16	yes
samuel	14	I	8	no attempt
gretta	14	III	8	yes
duncan	10	II	5	no attempt
slone	12	II	8	no attempt

The level of proficiency reached is based on Stoy's properties 1-3, respectively; a "+" indicates that the student spent a significant amount of time applying these concepts outside the specific domain of graphics programming. Time refers to the total number of hours that each student worked on this project. The final column notes whether students were able to predict the results of the same problematic expression described earlier (given to graduates and undergraduates). This question was administered as a test to students who reached level III; it was their first introduction to non-graphics Scheme code.

Although statistically the results of this study do not differ significantly from the results reported for MIT students in (Eisenberg, Resnick, & Turbak, 1987), it is important to note the differences in setting between the two situations. The subjects in the imaging curriculum had no special mathematical training and, in most cases, no similar academic experiences or background in which to frame their learning. The "successful" students who reached levels II and III were *not* MIT students: they were as young as 10 and 12 years of age.

Conversely, the students of this study had an array of imaging apparatus at their disposal which were unique to their situation. It is the claim of this work that presenting the student with an array of views on a concept (see also (Polya, 1945)Polya, 1945; Gardner, 1993) helps to build a more persistent and complete mental representation. Whether for the purpose of repeatedly encouraging visualization or for the purpose of catering to individual learning styles, the net result of a multiply represented system was a more exhaustive style of concept formation than that of a curriculum founded on a single representation.

In order to conjecture about the role of each specific tool used in this study, it is important to get a sense of where students' difficulties were concentrated. The results of the study reported here indicate (in agreement with the results of several pre-test studies (DiBiase, 1995b; Eisenberg, Resnick, & Turbak, 1987)) that Properties 1-3 of functional data objects represent strictly increasing levels of difficulty for students. Misunderstandings begin with Property 2 when students have to abstract functions in the same way that they manipulate variable numbers. By property 3, a number of interesting misunderstandings emerge.

Consider, for example, the difficulties (as noted above) with object anonymity in the case of function-generating functions. The literal existance of concrete objects like the felt pillows helps reify the notion of a nameless object. Students are able to see how these functions are built up through the `make-procedure` macro. In this way, students are able to view anonymous functions as reasonable -- even in the absence of a specific

label -- because they experience the function's genesis and composition. Another common misconception involving function-generating expressions is that students perceive them to be incompletely specified, referred to as *the missing variable phenomenon*. For instance, students of all the aforementioned studies who incorrectly answered the `(apply-to-5 create-subtracter)` example explained with overwhelming frequency that the interpretter could not complete this expression because "it doesn't know what `x` is yet." Misunderstandings about "missing variables" from lambda-style expressions are hindered by the process of using object manipulations. Unbound variables in a function-generating function are consistent with the real object generation process. Again, the visual aid of function and number manipulatives demonstrates that the function-generating function can legally execute without the information filled into those slots until a later time.

Unlike Gretta, some students were able to "graduate" from manipulative use and become fluent Scheme programmers. These two students (Aaron and Nate) successfully applied the higher-order function concept to a novel domain (Appendix 2 shows the worksheet students used to make observations about the slopes of functions). This satisfies one of the goals of this work -- that is, for students to proficiently apply the concept of higher-order function. Arguably, the imaging students surpassed this goal: 50% of the students were able to generalize Property 2 without explicit instruction. This ability to generate underlying concepts, I assert, shows greater cognitive depth than in cases where students are considered well versed simply on the basis of their ability to recognize or apply an underlying concept.

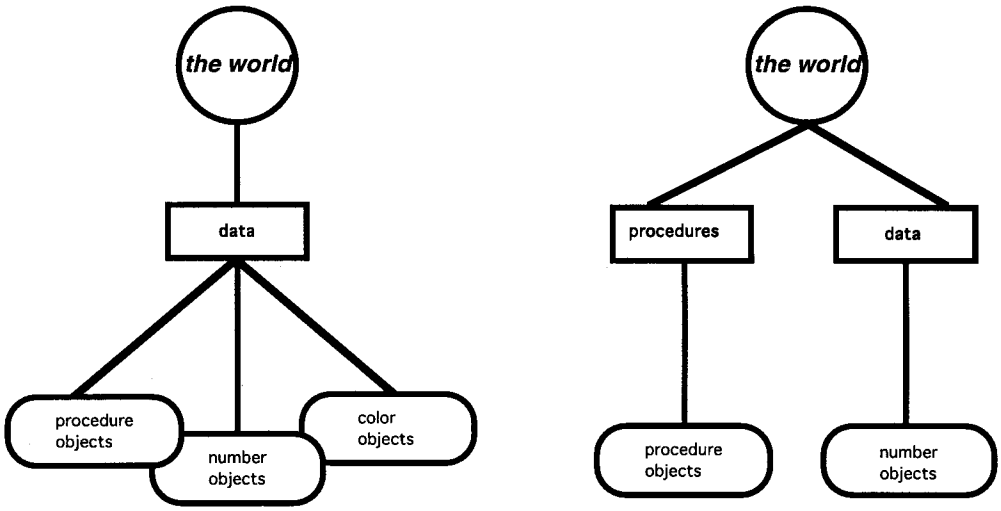
DISCUSSION

It is possible to use students specific ideas and misunderstandings to build a general ontological picture of their concept of function. Figure 7 shows a high level representation of two common cognitive models of functions. Ontology A depicts a consistent and cohesive view of data which I conjecture will lead its possessor to corresct problem solving. In this model, all entities are candidates to be data objects and all further classification occurs after that assumption. In the naive model (Figure 7, Model B), data and functions are two separate classes of things. In accordance with recent recommendations for a process-oriented system of mathematics education (Ponte, Matos, Matos, & Fernandes, 1991), the goal of the multi-modal imaging curriculum has been to rearrange the incorrect assumptions which lead to Model B in favor of the comprehensive view of data presented in Model A.

Empirical data for this report is derived from a case-study based approach to data collection. Therefore, statistics cited earlier only tell half the story about students' performance and do little to directly link the imaging curriculum to results. Instead, it is prudent to examine the actions and the language of the students who used the curriculum. Some of the cues may be given very explicitly by the students and others may be colored more by residual reflections of objects, for example:

- Students introspectively describe their own notions of functions as being very object-oriented.
- Even in the absence of the objects, students motion with their hands in a "container" shape as they describe the workings of the Scheme system.
- When students speak of procedure objects they use action-on-object verbs like throw, get, make-a-new, gives-back, etc.

These collections of identifying impressions -- referred to in this study as *imaging fingerprints* to reflect their relationship with the manipulatives -- characterize student's behavior in the absence of the tools themselves. Such information can be used to tell a far more exhaustive story than isolated statistical data alone. Imaging fingerprints, in conjunction with favorable test results, unite to paint a picture of students' ontologies of function which favors Model A.



Model A. Model B.

Figure 7 High Level Models of Functional Data

SUMMARY & CONCLUSIONS

The argument espoused in this paper for approaching the problem of higher-order function through multi-modal imagery building can be breifly summarized as follows:

- The concept of function -- particularly higher-order function -- is of paramount importance in mathematics and computer science;
- Students learning about functions -- particularly higher-order functions -- chronically exhibit error-ridden learning behavior;
- The core difficulty with functions as data objects is a specific instantiation of a more general problem with abstraction;
- Imagery tools can be usefully employed to help students reify abstraction;
- A more diverse tool set can lead to broader concept accessibility for a range of learners.

REFERENCES

(1958). The Concise Oxford Dictionary. London: Oxford University Press.

Aleksandrov, A. D., Kolmogorov, A. N., & Lavrent'ev, M. A. (Ed.). (1963). Mathematics: Its Content, Methods, and Meaning. Cambridge: MIT Press.

Allen, J. (1987). Natural Language Understanding. Menlo Park: Benjamin-Cummings.

Ayer, T., Davis, G., Dubinsky, E., & Lewin, P. (1993). Computer Experiences in Learning Composition of Functions No. Clarkson University.

Babbage, C. (1842). Notes on the Analytical Engine. In P. M. a. E. Morrison (Eds.), Charles Babbage and His Calculating Engines: Selected Writings by Charles Babbage and Others (pp. 225-295). New York: Dover Publications, Inc.

Breidenbach, D., Dubinsky, E., Hawks, J., & Nichols, D. (1992). Development of the Process Conception of Function. Educational Studies in Mathematics, 23, 247-285.

Chi, M. T. H., Slotta, J. D., & Leeuw, N. d. (1994). From Things to Processes: A Theory of Conceptual Change for Learning Science Concepts. Learning and Instruction, 4, 27-43.

Cuoco, A. (1993b). Constructing Functions from Algebra Word Problems. Education Development Center Report.

Cuoco, A. (1995). Computational Media to Support the Learning and Use of Functions. In A. A. diSessa, C. Hoyles, & R. Noss (Eds.), Computer and Exploratory Learning Berlin: Springer.

DiBiase, J. (1995a) Building Curricula to Shape Cognitive Models: A Case Study of Functions as Data Objects. Ph.D. Thesis, University of Colorado, Boulder.

DiBiase, J. (1995b). Examining the Difficulty with Thinking of Functions as Data Objects: Misconceptions of Higher-Order Functions. No. CSCI-791-95, University of Colorado, Boulder.

Eisenberg, M. (1991). Programmable Applications: Interpreter Meets Interface. SIGCHI Bulletin, 27(2), 68-83.

Eisenberg, M., Clinger, W., Harheimer, A., & Abelson, H. (1990). Programming in MacScheme. San Francisco, CA: The Scientific Press.

Eisenberg, M., Resnick, M., & Turbak, F. (1987). Understanding Procedures as Objects. In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), Empirical Studies of Programmers: Second Workshop (pp. 14-32). Norwood, NJ: Ablex Publishing Corp.

Engelkamp, J. (1988). Cognitive and Neuropsychological Approaches to Mental Imagery

Engelkamp, J., & Zimmer, H. (1990). Imagery and Action: Differential Encoding of Verbs and Nouns. In P. Hampson, D. Marks & J. Richardson (Eds.), Imagery: Current Developments London: Routledge.

Gardner, H. (1993). Multiple Intelligences : the Theory in Practice. New York: Basic Books.

Gödel, K. (1931). On Formally Undecidable Propositions of Principia Mathematica and Related Systems. New York: Dover.

Harel, G., & Dubinsky, E. (Ed.). (1992). The Concept of Function: Aspects of Epistemology and Pedagogy.

Harel, I., & Papert, S. (Ed.). (1991). Constructionism: Research Reports and Essays. Norwood, N.J.: Ablex.

Hughes, M. (1986). Children and Number: Difficulties in Learning Mathematics. Oxford: Basil Blackwell.

Kaufmann, G. (1988a). Mental Imagery and Problem Solving. In M. Denis, J. Engelkamp, & J. T. E. Richardson (Eds.), Cognitive and Neuropsychological Approaches to Mental Imagery Dordrecht: Martinus Nijhoff.

Kaufmann, G. (1988b). Mental Imagery in Problem Solving. In A. A. Sheikh (Eds.), International Review of Mental Imagery New York: Human Sciences Press.

Kaufmann, G. (1990). Imagery Effects on Problem Solving. In In P. Hampson, D. Marks & J. Richardson (Eds.), Imagery: Current Developments London: Routledge.

Kieran, C., Garaicon, M., Lee, L., & Boileau, A. (1993). Technology in the Learning of Functions: Process to Object? In Psychology of Mathematics Education, 15 . Asilomar Conference Center:

Kosslyn, S. (1980). Image and Mind. Cambridge, MA: Harvard University Press.

March, R. (1977). Georges Cuisienaire and his Rainbow Rods. Learning, 6(3), 85-86.

Marks, D. F. (1972). Individual differences in the vividness of visual imagery and t heir effect on function. In P. Sheehan (Eds.), The Function and Nature of Imagery New York: Academic Press.

Marks, D. F. (1990). On the relationship between imagery, body, and mind. In P. Hampson, D. Marks & J. Richardson (Eds.), Imagery: Current Developments London: Routledge.

Marschark, J. C. Y. a. M. (1988). Imagery and Children's Learning. In A. A. Sheikh (Eds.), International Review of Mental Imagery New York: Human Sciences Press.

Montessori, M. (1956). The Child in The Family. New York: Avon Books.

Paivio, A. (1971). Imagery and Language. In S. J. Segal (Eds.), Imagery: Current Cognitive Approaches New York: Academic Press.

Paivio, A. (1988). Basic Puzzles In Imagery Research. In M. Denis, J. Engelkamp, & J. T. E. Richardson (Eds.), Cognitive and Neuropsychological Approaches to Mental Imagery Dordrecht: Martinus Nijhoff.

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

Polya, G. (1945). How to Solve it? Princeton, NJ: PUP.

Ponte, J. P., Matos, J. F., Matos, J. M., & Fernandes, D. (Ed.). (1991). Mathematical Problem Solving and New Information Technologies: Research in Contexts of Practice. Berlin: Springer-Verlag.

Pylyshyn, Z. (1981). Imagery and Artificial Intelligence. In N. Block (Eds.), Readings in the Philosophy of Psychology Cambridge, MA: Harvard University Press.

Sfard, A. (1992). The Case of Function. In G. H. a. E. Dubinsky (Eds.), The Concept of Function Mathmatical Association of America.

Sheehan, P. (1972). Imagery and Unexpected Recall. In P. Sheehan (Eds.), The Function and Nature of Imagery New York: Academic Press.

Thyer, D., & Maggs, J. (1971). Teaching Mathematics to Young Children. London: Holt, Reinhart and Winston.

Turing, A. (1937). On Computable Numbers, with an Application to the Entscheidungsproblem. In Proceedings of the London Mathematical Society, 42 (pp. 433-460).

Turing, A. M. (1950). Computing Machinery and Intelligence. Mind, 59, 433-460.

Tye, M. (1991). The Imagery Debate. Cambridge: MIT Press.

Von-Neumann, J. (Ed.). (1945). First Draft of a Report on the EDVAC. New York: Springer-Verlag.

Waterfield, R. (Ed.). (1974). Plato: Republic. Oxford: Oxford University Press.

Wilensky, R. (1986). Common LISPcraft. New York: Norton.

Winston, P. H., & Horn, B. K. P. (1989). LISP (3rd ed.). New York: Addison-Wesley.

APPENDIX A

APPENDIX B

APPENDIX A

It would be posssible to write a procedure which squared numbers:

```
>>> (define square
        (make-procedure
            (x)
            (* x x)))

>>> (square 5)
25
```

And likewise, we could cube numbers:

```
>>> (define cube
        (make-procedure
            (x)
            (* (* x x) x)))

>>> (cube 2)
8
```

We could go on like this and write a procedure which raised numbers to the 4th power, and another for the 5th power, and so on. Or, we could save some time by writing a procedure which generated any power procedure that we wanted. So we would use it like this:

```
>>> (define cube (make-power-procedure 3))

>>> (cube 2)
8
```

Write make-power-procedure.

Give some examples of ways that you would use it. For example, how can I take the 10th power of the number 3 without ever defining a 10th power procedure into the table?

```
>>> (define subtract-from-5
      (make-procedure (x) (- 5 x)))
```

What does this procedure do?
It decreases the number 5 by an input (x),

Give an example of how you would use it and what the result would be of your example.

```
>>> (subtract-from-5 10)
    -5
```

```
>>> (define apply-to-5
     (make-procedure
          (f)
          (f 5)))
```

What does this procedure do?
It takes a procedure (f) as an input and inputs 5 to that procedure.

Give some example uses and results.

```
>>> (apply-to-5 (subtract-from-5))
    0
>>> (apply-to-5 (make-power 3))
    125
```

```
>>> (define make-doubler
       (make-procedure
          ()
          (make-procedure (x) (* 2 x))))
```

Then what would happen if we typed in the following expressions:

```
>>> ((make-doubler) 5)
    10

>>> (make-doubler)
    <PROCEDURE>

>>> make-doubler
    ?
```

subtractor

```
>>> (define create-subtracter
        (make-procedure
            (n)
            (make-procedure (x) (- x n))))
```

What does create subtracter do? What does it take as arguments and what does it return?

Create subtractor makes a procedure that outputs "subtract n from me". It takes the argument n. It returns a procedure

sorry →

What would be the result of the following expression:

```
>>> (create-subtracter 3)
```

<PROCEDURE>

What do the following expressions do?

```
>>> (define subtract-3 (create-subtracter 3))
```

Defines subtract-3 as a program that subtracts 3 from numbers.

```
>>> (subtract-3 10)
```

Subtracts 3 from 10.
Outputs 7

What happens when we call apply-to-5 on create-subtracter?

```
>>> (apply-to-5 create-subtracter)
```

Creates a program that subtracts 5 from numbers.

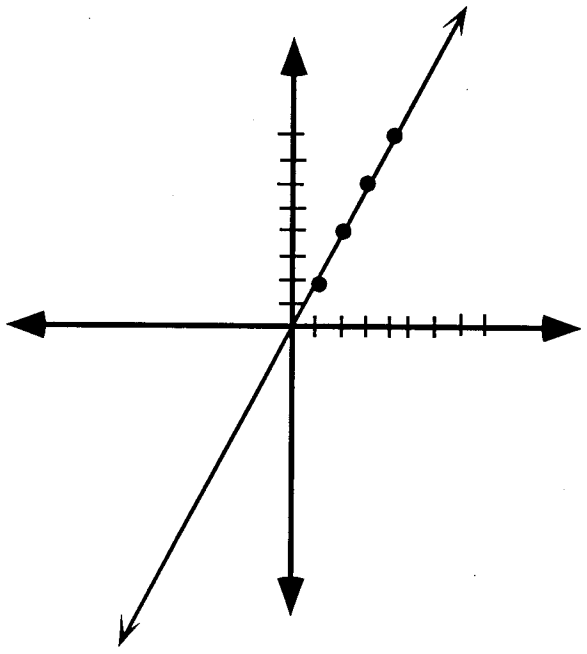
APPENDIX B

Say we have the following functions which operate on numbers:

```
>>> (double 2)
4
>>> (triple 2)
6
```

1. Write `double` and `triple`.

Remember from math that we can represent functions as graphs. So, `double` could also be written y=2x or f(x) = 2x. We could graph it as follows:



Recall also that the slope of a line refers to how much the function changes value over a given interval. So, between x=0 and x=1, the function's slope is f(1) - f(0), or 2-0, or 2.

2. What is the slope between x=1 and x=2?
between x=2 and x=3?

Different functions may have different slopes.

3. What is the slope of the `double` function?
of the `triple` function?

We would like to write a procedure which lets us evaluate slope for different functions. It would work like this:

```
>>> (define double-slope-function (slope double))
>>> (double-slope-function 0)
2
>>> (double-slope-function 1)
2
```

The interval used to calculate the slope is defined by the input and the input+1. [so, `(double-slope-function 2)` calculates between 2 and 3]

4. Write `slope`.

5. Use slope on the `double` and `triple` functions. Predict what the result should be first.

6. Write a `quadruple` procedure. What should its slope be?

7. Write a `square` procedure which returns the 2nd power of numbers. [`(square 3) = 9`, for example]

8. Use your slope function to make a chart for `square`. Do you see any patterns.

x-interval	slope
0 1	
1 2	
2 3	
3 4	

9. Write a `cube` function and make a similar chart. Try to predict the results beforehand.