“My criterion is: Is it a Boolean?”: A card-sort elicitation of students’ knowledge of programming constructs

Marian Petre Sally Fincher Josh Tenenberg Richard Anderson
Ruth Anderson Dennis Bouvier Sue Fitzgerald Alicia Gutschow
Susan Haller Matthew Jadud Gary Lewandowski Raymond Lister
Renée McCauley, John McTaggart Briana Morrison Laurie Murphy
Christine Prasad Brad Richards Kate Sanders Terry Scott
Dermot Shinners-Kennedy Lynda Thomas Suzanne Westbrook
Carol Zander

June 2003

1 Introduction

This study examined the way in which students understand programming concepts. If we can understand the nature and structure of students’ knowledge about programming constructs, then we may be able to use that understanding to help them learn. Educators know which concepts they teach, but not what students internalise about those concepts, nor what conceptual structures students build from them.

Specifically, we wanted to know whether students have meanings for:

- individual programming concepts,
- groups of related concepts, and
- relationships among groups of related concepts.

For example, we might ask if students have a meaning for “tree.” What concepts do they group “tree” with, and what name do they give the group? If they group “tree” with “list” and “array” and call the group “data structures,” what other groups of concepts do they associate with “data structures”?

Because of the diversity of researcher and student population, it was important to choose a method not constrained by any programming task or the syntax of a particular programming language. We adopted a methodology, card sorting, from knowledge acquisition to elicit each participant’s knowledge structure (mental model, conceptual model) of programming concepts.

We made several key assumptions:

- First, the way in which a subject organizes concepts in a card sort reflects the subject’s mental representation of those concepts.
- Second, by putting a card into a meaningful category, subjects demonstrate that the concept on the card has some meaning for them.
- Third, by putting a card into a category, subjects indicate what the category and the related criterion mean to them.

By examining the ways in which students sorted the cards, we hoped to gain insight into the conceptual structure of their knowledge about programming constructs and program construction.

Our initial analysis of the data focused on the following questions:

- Do students and educators organize concepts differently, and if so, how?
- Are there differences between male and female students?

- Are there differences between students based on the programming languages they know?
- Are there significant indicators in students' use of "don't know" categories?

In Section 2 of this paper, we review the related work on card sorts and novice understanding of programs. In Section 3, we discuss our research methodology. In Section 4, we present our analysis, and we discuss our results and directions for future work in Section 5.

2 Related Work

2.1 Conceptual structure

The goal of this study was to investigate the conceptual knowledge of "first-competency" programmers. In particular, the study investigated the meaning students attach to programming concepts. There is evidence to suggest that the way in which subjects organize concepts reflects their mental representation of the way these concepts are related. Adelson gave novice and expert programmers randomly ordered lines of computer code and observed how they recalled the code and in what proximity the lines were recalled. [1] The proximity of the lines' recall was evidence that the subjects were imposing their own structure on the unstructured data.

Different populations exhibit different concept organization structures. Various studies contribute to the notion that developing expertise is reflected in knowledge consolidation, and that consolidation of knowledge into meaningful conceptual structures is a reflection of "deep learning", characterised by abstracted meta-knowledge. For an overview, see e.g. Eysenck and Keane [8] and Marton and Säljö [10]. Adelson [1] noted that experts had more consistent subjective organization than novices. One consistent result in the novice-expert literature is that experts organize or "chunk" information differently from novices: they form abstractions based on deep (semantic) characteristics rather than on surface (syntactic) characteristics. Allwood [3] noted that "novices used general memory strategies while experts used a more specific strategy" and they "showed large variation in their organization of the investigated concepts." Their findings were similar to those of Chi et al. [4], which indicated that novices sort information on the basis of surface features whereas experts sort on the basis of underlying structure.

2.2 Elicitation of conceptual structure

Elicitation of internal conceptual structures is problematic because it requires plausible, observable intermediate representation. One mediating activity described in the literature is card sorting. In a card sort, subjects are presented with a set of cards, with each card having a single picture, name of a concept or a short description written on it. Subjects are asked to sort all of the cards into different groups, naming both the groups and the basis or *criterion* along which items are sorted. Subjects are then asked to repeat the sort – using a different criterion – and then to keep on sorting until they have run out of criteria.

For example, if the task was sorting pictures of different types of house, a subject might sort them into groups "brick," "stone," "wood," etc., with the criterion being "main material of construction." The second time, the subject might divide the cards into groups called "one," "two," and "three," with the criterion being "number of floors in each building."

Card sorting (Rugg & McGeorge, [11]) has been used to elicit information on internal representations of concepts. Davies, Gilmore and Green ([5]) used card sorting of code fragments to obtain expert and novice computer programmer's knowledge about relationships among program components. Subjects were asked to sort the cards into categories that had meaning to them and to justify their sort. Given Adelson's evidence [1], Davies et al ([5]) expected experts to base their categorizations on objects and inheritance relationships and novices on syntactic elements. Instead, results indicated that experts mainly based their classifications on the functional relationships between code fragments while novices mainly derived their classifications from object-based categorizations. Davies et al. also predicted more consistent classifications from experts and more arbitrary and idiosyncratic sorts from novices: the contrary was true.

There is a tradition of using card sorting as a way of eliciting conceptual structures, in the general literature and more specifically within the discipline of software. The relationships exposed

1	function	10	scope	19	type
2	method	11	list	20	loop
3	procedure	12	recursion	21	expression
4	dependency	13	choice	22	tree
5	object	14	state	23	thread
6	decomposition	15	encapsulation	24	iteration
7	abstraction	16	parameter	25	array
8	if-then-else	17	variable	26	event
9	boolean	18	constant		

Figure 1: Stimuli used in card sort task.

by categorisation tasks are taken to reflect relationships in the subjects' internal representations and hence to lead to a model of their internal representations. [1]

3 Study Methodology

The study was unusual in its scope, involving more than twenty researchers from four continents and six countries. Each researcher was an experienced college-level computer science educator. The researchers' institutions included public and private institutions that used a variety of approaches to teaching programming. Each researcher collected data from his or her own institution against a standard protocol; the combined corpus included 275 subjects.

The primary method, described in more detail below, was a repeated single-criterion card sort [11] designed to elicit subjects' knowledge of programming concepts.

3.1 Subjects

The 275 subjects included computer science students and faculty at twenty-one different colleges and universities in Australia, Barbados, Ireland, New Zealand, the United Kingdom, and the United States. Thirty-two were educators, and 243 were students. Of the students, 185 were male and 58 were female.

The student subjects were "first competency programmers," that is, they were selected at the point in their curriculum where they were considered capable of solving one of a set of programming problems drawn from the McCracken test set. [7] Their performance in computer programming courses varied widely. The faculty subjects included educators from the same institutions who had experience teaching introductory programming (though they may not have taught any of the student subjects in this experiment).

3.2 Stimuli

We developed a set of twenty-six minimalist one-word prompts for programming concepts (see Figure 1). The concepts were general ones, such as "tree" or "variable." The prompts were drawn from programming textbooks, from papers on program categorizations, and from lists generated by programming experts and programming educators.

This stimulus set was first reviewed by two programming educators and then tested with seven participants from two locations including three first-year college programmers, and two computing researchers. The stimulus set proved usable by all participants. Additional benefits include spontaneous participant viewpoints and manageable data. Further, preliminary cluster analysis suggested potentially interesting novice vs. expert differences and indications of misconceptions.

3.3 Interview Procedure

3.3.1 Preliminary procedure

Subjects were asked to complete a background questionnaire (see sample questionnaire in Appendix A), and to sign a consent form after reading details of the experiment and discussing them with the researcher. Subjects were then given a description of the card-sort task (Appendix B). Some researchers demonstrated a "card sort" using simple alternative stimuli.

3.3.2 Card-sort task

We gave the subjects the set of twenty-six index cards, each labelled with a programming concept, and asked them to sort the cards into categories using a single criterion. To avoid imposing our own criteria or categories, we asked the subjects to provide them. Participants were asked to provide names for each group (category), and for the overall criterion by which the cards were sorted. This information was recorded, along with a list of the cards contained in each category. Subjects were asked to perform sorts repeatedly until they were unable (or unwilling) to carry out additional sorts.

When the subjects indicated they could think of no additional sorts, they were presented with a triadic prompt-a group of three cards-and asked to sort them into two categories. If they were able to, they were then asked to organize the remaining cards according to the same criterion. The researcher recorded which cards were used for the triadic prompt, typically object, scope, encapsulation. If the initial triadic prompt was unsuccessful, some researchers tried a second time.

3.4 Data Collection

3.4.1 Background data

Age, gender, and programming language familiarity were collected for each subject. For student participants, grades in programming courses were also recorded. Some institutions collected additional data, which included whether the student was full-time or part-time, whether they attended day or night classes, if the student held an external job, and if so, how many hours were worked each week, and if the job was in the computer industry, the language used in their first and second programming courses, and at what age they began to program.

3.4.2 Task data

Criterion names and category names were recorded verbatim. The cards in each category were recorded by number. During sorting, some researchers also recorded information on subjects' card-handling behavior and, where appropriate, observational notes and ephemeral sorts.

4 Analysis and Discussion

4.1 Data Analysis Techniques

We used several techniques to summarize and link the data. We entered card-sort data into an Excel table and uploaded this table (see example below) into a project database to allow us to perform automated analysis on the entire dataset and subpopulations of it. We summarized the card-sort data for each subject in a separate spreadsheet.

A portion of the spreadsheet for one subject is given in Figure 2. The leftmost column contains the criterion for each sort (eight criteria for this student), with the first criterion being "tangible and abstract." The next column lists the categories in each sort. In the first sort there are two categories "tangible" and "abstract," To the right of that column there are columns representing the cards. In the complete spreadsheet, there are 26 such columns, one column for each card. In the example below, however, not all columns are shown. An "x" in a column indicates that the card in that column was placed into the category listed on that row. For example, here the terms "function" and "procedure" were grouped in the same category (co-occurred) in all eight sorts, and terms "state" and "event" were grouped in seven of the eight sorts.

A number of mechanisms were used to assist analysis, collected into a project toolkit, some exploratory (to help us form more focused questions about the data), and some summative and comparative (to provide numerical tools for characterisation):

- Analysis of background characteristics by sub-population
- Verbatim analysis (agreement on actual names of criteria and categories)

Criteria name	*Category name*	*function*	*procedure*	*state*	*event*
tangible and abstract	tangible	x	x		
	abstract			x	x
principles	principles				
	not principles	x	x	x	x
data	places to put data				
	types of grouped data				
	types of primitive data				
	everything else	x	x	x	x
programming structures	definitely programming structures	x	x		
	might be programming structures			x	x
	not programming structures				
approaches	my object oriented world				x
	my structured world	x	x		
	overlap			x	
OO programming	pure OO programming				
	not OO programming	x	x	x	x
control structures	control structures				
	everything else	x	x	x	x
modularisation	modularisation	x	x		
	everything else			x	x

Figure 2: Excerpt from one subject's data.

- Gist analysis on names (agreement on the meaning of criteria and categories, despite different verbatim naming)
- Gist analysis on cards (identifying same or similar grouping of cards, regardless of naming)
- Co-occurrence matrices (identifying the frequency with which cards appeared together) for individuals and for sub-populations
- Dendrograms generated from matrices summarizing individuals' sorts
- Distance analysis tool based on edit distance

4.1.1 Analysis of background characteristics by sub-population

An initial analysis of the data was made to determine certain "demographics" of the overall population such as average age over all subjects, gender breakdown, percentage of subjects that were student versus expert, breakdown by academic performance (students only), and breakdown by familiarity with specific programming languages (Java, Scheme, C++, C, Pascal, Ada, and others).

4.1.2 Verbatim analysis

Verbatim analysis across criteria and category names was automated as part of the project toolkit.

4.1.3 Co-occurrence matrices and gist analysis on cards

We constructed a co-occurrence matrix for each subject. The matrix records the frequency with which each pair of cards are grouped in the same category over all sorts. With respect to subjects' sorts, we calculated average number of sorts per subject and average number of categories per sort criterion. For each category of each sort, we linked to other subjects with exactly the same cards given in a category. We distinguished from this but also included links to categories with a one-card difference (one more card, one less card, or one different card). These links were summarized in pair-wise frequency tables which could be generated within a sub-population.

4.1.4 Gist analysis on names

We also looked for sort criteria that had similar meaning (or gist). For example, we might consider a sort criterion such as "object-oriented concepts" to have the same gist as a sort criterion called "related to object-oriented." Similarly, "loop," "iterative," "repetition," and "looping flow" might all be considered to have the same gist. We used a tool to help identify criteria that possibly had the same gist by comparing keyword pairs in the criteria names. For example, from the criterion name "Concepts According to How Difficult to Use and Learn" we extracted open terms - nouns and verbs: "concepts," "use," and "learn," formed sets of size two concepts, use, concepts, learn, use, learn and compared these to other criterion names. This worked as an initial search tool for gist among criterion names; however, it would not have found the pair of criteria above, namely "programming structures" and "imagining that I want to write a program." To find these matches we worked as a group.

4.1.5 Dendrograms

Using the programs EZCalc and EZSort [6], a hierarchical cluster analysis was computed on a distance matrix for each subject. Initially we generated the distance matrices using EZCalc and EZSort; latterly we generated four distance matrices: using Manhattan distance and Euclidean distance, and using Simple and Jaccard's similarity measures subtracted from one to yield a distance measure. From each of these matrices, we generated dendrograms using simple (nearest neighbour), complete (maximizing distance between clusters), and Ward's (minimizing intra-cluster distance) methods of clustering [2]. These dendrograms were used as exploratory visualizations to help us focus our questions.

4.1.6 Interpretation of Dendrograms

A dendrogram is a hierarchical clustering of sorting data. Consider the terms "function" and "procedure" in the dendrogram below, the lines emanating from each of those two terms are joined at the vertical line labeled "0" indicating that these cards were always placed in the same category. Recall that these terms co-occurred eight out of eight times in the spreadsheet above. In contrast, the lines emanating from the terms "state" and "event" join further to the right, indicating that although they are frequently associated, they are not always associated. Recall that these terms co-occurred seven out of eight times in the spreadsheet. The lines emanating from the terms "method" and "object" intersect even further to the right indicating that they co-occurred less often than "state" and "event." Moving from left to right in the dendrogram, terms are combined into larger and larger clusters. The more often terms occur together, the further to the left the connection between them.

4.1.7 Distance Analysis Tool

Another approach to analysing the card-sort data is to look at the distance between individual sorts. This can be used to look for similar categorization across individuals or as the basis for a clustering analysis. Our definition of distance is based on the notion of edit distance, which counts the number of primitive operations to convert one string to another. In this case the primitive operation is to move a card between piles (or to a new pile). The distance between two sorts is defined to be the minimum number of moves to convert one to another. It can be shown that this distance function is a metric so that it can be used as a basis for clustering analysis. The distance can be computed by computing by finding maximum weight matching between the two sorts. A matching between two sorts is a correspondence of piles, if the number of piles is not equal, then some piles will not have corresponding piles in the other set. The weight of a matching is the number of common elements in corresponding piles. The maximum weight matching gives the closest correspondence between two sorts. Since the number of cards was 26, a matching of weight 26 indicated perfect correspondence.

We developed a tool to compute the distance between sorts. This was a stand alone application written in C#. The core of the application was an algorithm for computing the maximum weight matching in a complete bipartite graph. The tool allowed pair-wise comparison of sort, as well as comparing a sort against all other sorts, and comparing all pairs of sorts. Comparing a sort against all other sorts allowed identifying the closest neighbors of a sort. The all pairs comparison was run

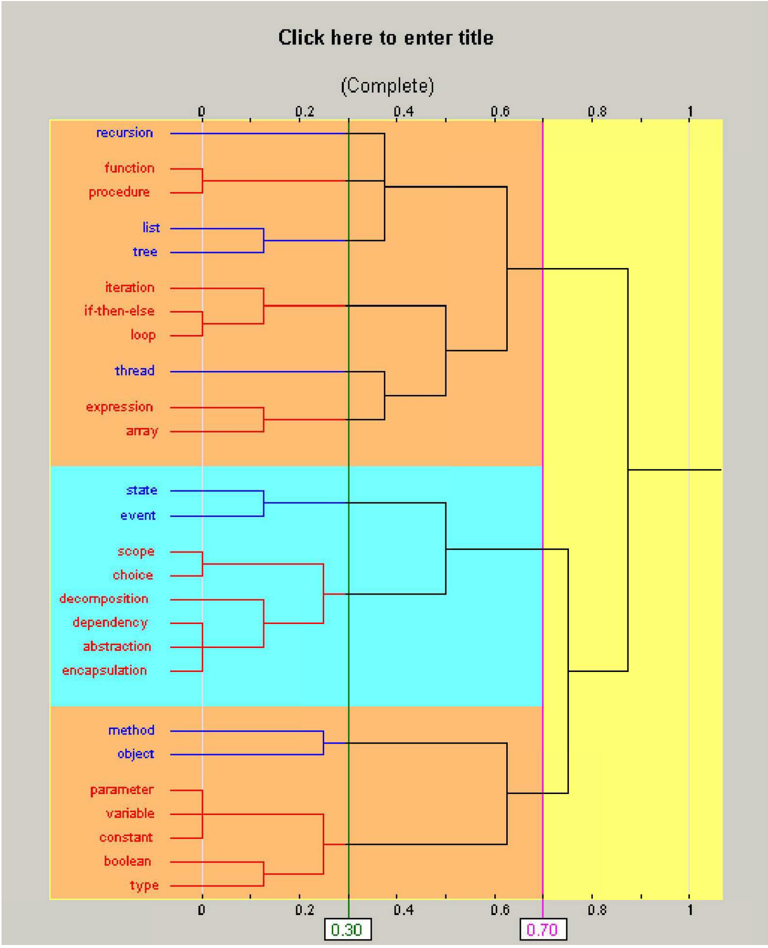

Figure 3: Moving left to right in the dendrogram, term clusters are repeatedly joined based on their "cluster" co-occurrence.

over the full set of 1198 sorts. We discovered 21 pairs of sorts with a matching of weight 26 (perfect correspondence), 48 pairs of weight 25 (distance one), and 163 pairs of weight 24 (distance 2).

4.2 Difficulty of Learning, Order of Learning

The concepts of "order of learning" and "level of difficulty" appeared frequently as sort criteria. Under "order of learning," we included all the criteria that represented the sequential learning process, such as "order of learning," "order things were presented," "order things should have been presented," and "order of presentation in a textbook." Under "level of difficulty" we included criteria such as "simpleness," "complexity," and "more versus less advanced." Within each of the "order of learning" criteria, we identified categories that corresponded to "early" and "late," and within each of the "level of difficulty" criteria we identified categories that corresponded to "easy" and "difficult." We then determined the frequencies with which stimuli appeared in each of these categories.

All of the stimuli categorized most frequently as "easy" were also categorized most frequently as "early." As shown in Figure 4 these include concepts such as "variable," "constant," and "function."

Similarly, stimuli that were most frequently categorized as "difficult" were also categorized most frequently as "late" in the learning sequence. As shown in Figure 5, these included concepts such as "encapsulation," "decomposition," and "abstraction."

There is no inverse relationship. Nothing that appears in the "late" category appears in the "easy" category, and nothing that appears in the "early" category appears in the "difficult" category. None of the students who performed these sorts placed "decomposition," "encapsulation," or "tree"

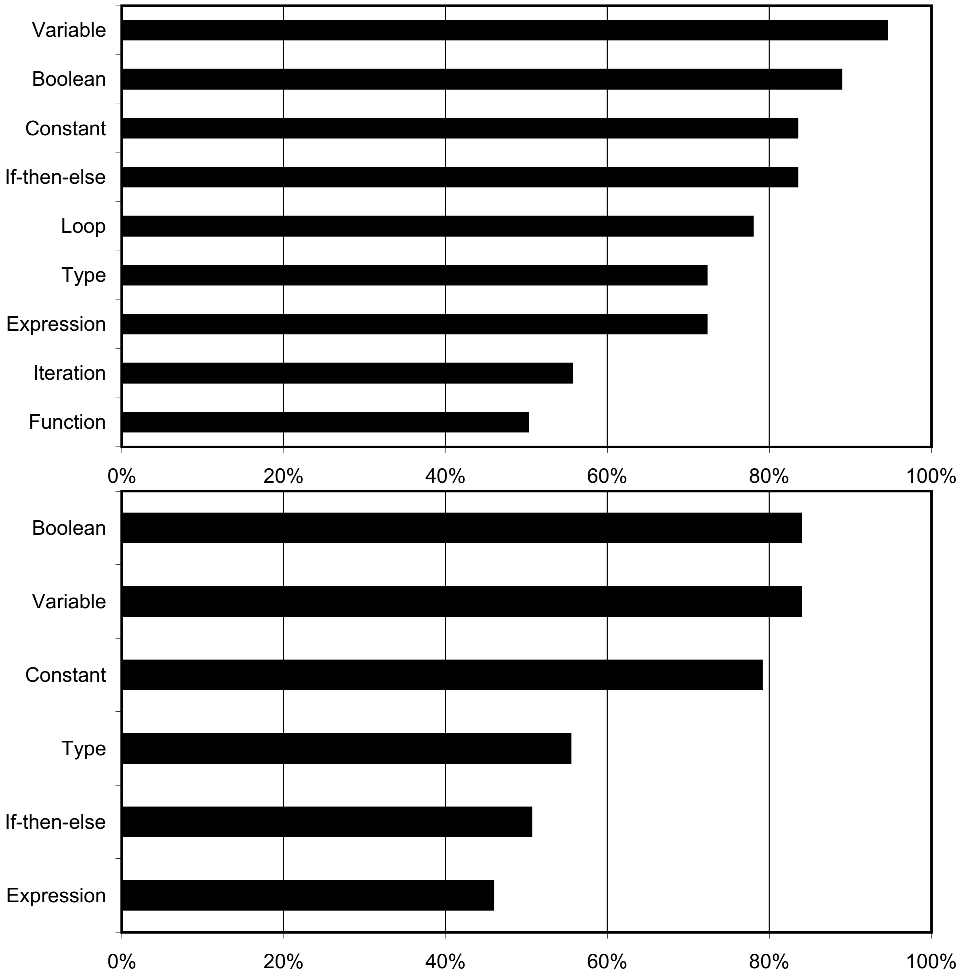

Figure 4: Stimuli categorized as "early" or "easy."

in the "early" category, and none of them placed "function," "procedure," "if-then-else," "boolean," "variable," "constant," "loop," or "expression" in the "late" category. None of the students placed "dependency" in the "easy" category, and none of them placed "boolean" or "variable" in the "difficult" category.

The students who performed these sorts appear to be representative of the student population as a whole. There were eighteen of these students. As shown in Figure 6, the percentage of women in these subpopulations is comparable to that in the general student population. We found no differences with regard to age, performance level, or programming-language background between these students and the general student population.

In addition to the eighteen students, five educators performed sorts using "level of difficulty" and/or "order of learning" criteria. The categorizations used by students were similar to, but not identical to the categorizations used by educators. The stimuli the majority of the students categorized as "easy" were also categorized as "easy" by a majority of the educator subpopulation. However only two of the stimuli, "recursion" and "tree," that were most often categorized as "difficult" by students were also categorized as "difficult" by educators. All of the stimuli most often categorized as "early" by the students were most often categorized as "early" by the educators, with the exception of "expression" and "type." "Expression" and "type" were categorized by students as "early," but not by educators.

This data raises the question of whether material that is presented early is generally perceived as easy by the students, perhaps because they have more time to absorb it, perhaps because it is emphasized by the instructor. To answer this question, however, we need to obtain data from students to whom concepts have been presented in different orders.

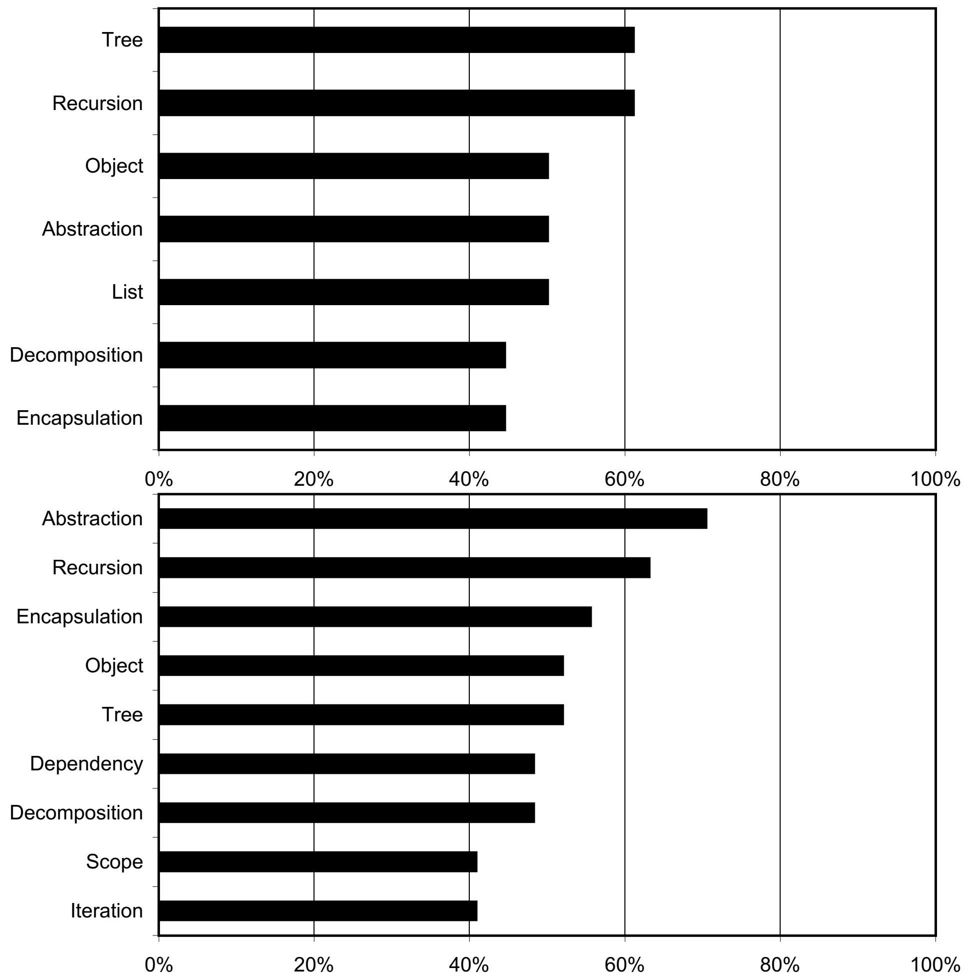

Figure 5: Stimuli categorized as "late" or "difficult."

In particular, it would be interesting to obtain data from students who are introduced to abstract concepts such as "encapsulation" early. The "late" and "difficult" concepts in our data are also the more abstract concepts. As a result, the apparent correlation between order of learning and level of difficulty might be due to a general tendency to find concrete concepts easier.

4.3 Gender

Are there differences in the way in which men and women students organize their knowledge of programming concepts? In order to answer this question, we compared a number of factors:

- the number of sorts performed
- the average number of categories per sort
- the use of binary sorts (that is, the sorts in which there were precisely two categories)
- the use of oppositional criteria (criteria where the categories can be ordered along a scale)
- the use of a "don't know" category.

In the quantitative factors we examined, there was little difference between male and female students, as shown in Figure 7. The average number of sorts is similar for both genders (except for the over-40 category). On average, women have more categories per sort (4.4 vs. 4.0). Women are also slightly more likely to use both binary sorts (41.4% vs. 40%) and sorts involving oppositional criteria (17.2% vs. 16.2%).

	No. of subjects who are		*% of total subjects who are*	
	women	*men*	*women*	*men*
Total student population	58	185	24%	76%
"Order of learning" subpopulation	4	14	22%	78%
"Level of difficulty" subpopulation	5	23	18%	82%

Figure 6: "Order of learning" and "level of difficulty" subpopulations compared with the general student population.

	Men	*Women*	*Total*
Number of students	185	58	243
Number of sorts	831	258	1089
Number of categories	3284	1131	4415
Number of students who used binary sorts	74	24	98
Number of students who used oppositional criteria	30	10	40
Number of oppositional criteria	43	14	57
Average number of sorts	4.5	4.4	4.5
Average categories / sort	4.0	4.4	4.1
Percent who used binary sorts	40	41.4	40.3
Percent who used oppositional criteria	16.2	17.2	16.5

Figure 7: Breakdown by gender.

Based on a preliminary analysis, the qualitative data also revealed little difference between male and female students. We identified several groups of criteria that had approximately the same meaning, as shown in Figure 8. Of this initial list, we focus on four: the creative analogies, the oppositional criteria, the emotional response, and the no-name criteria. Notably, the difference between males and females in each of these areas was quite small.

4.3.1 Creative analogies

Four criteria made analogies to situations outside of computer science. These sorts are summarized in Figure 9.

These analogies were made by four different students, two male and two female. One female student related the concepts to painting, comparing some to the palette and others to the finished painting. The other female student used a sports analogy, comparing concepts to players, formations, and coaching. One male student compared programming structures to a Russian doll, apparently thinking of nested programming structures. Finally, the second male student suggested an analogy between the concepts and the tools and materials used in construction.

What can we conclude? The numbers of male and female students suggesting these analogies were equal. More significantly, both were very small compared to the total number of sorts. We cannot conclude anything significant about gender differences from these examples, but it does seem clear that very few of our subjects use concepts from outside of computer science to organize their knowledge about programming.

4.3.2 Scalar criteria

"Scalar criteria" are criteria that order the concepts along a scale from one extreme to another: objects vs. functions, concrete vs. abstract, design vs. implementation, compile-time vs. runtime, and so forth. Scalar criteria may or may not result in binary sorts. For example, a student might classify all the cards as either "concrete" or "abstract," or might identify several levels of abstraction. Several subjects gave more than one scalar criterion, so the number of subjects is consistently lower than the number of criteria.

We conjectured that male students would have more sorts that fell into this group, and surprisingly, that turned out to be false. Of the sorts done by male students, 43 out of 831, or 5%, contained

Abstract/abstraction	Behaviour
Change	Code
Complexity	Concepts
Concrete	Control/control structures/control flow
Creative analogies	Datatype/data/data structure/variables
Dependency	Design
Don'tknow/unknown/other/not applicable	Emotional reaction
Functions	GUI or event-oriented
Hierarchy	Ideas
Information hiding	Judgemental
Level of difficulty/ease/simplicity	Lifetime
Natural language related	No name
Object-oriented	Oppositional
Parts of a program	Practicality
Programming language related	Programming lifecycle
Programming paradigms	Relationship

Figure 8: Possible "gists" found in the card-sort criteria.

Criterion	*Gender of subject*	*Related categories*	*Analogy to*
structure - a sliding scale - the Russian doll effect	male	Most general, 2nd most general, 3nd most general, 4th most general, not applicable	the arts
Elements of construction	male	construction materials	construction
Painting analogy	female	palette, finished product, don't know	the arts
Actors and manipulators	female	players, formations, coaching, unsure	sports

Figure 9: Criteria involving analogies to a situation outside computer science.

scalar criteria. Of the sorts given by female students, 14 out of 258 contained scalar criteria, which also rounds to 5%. Considered as a percentage of subjects, the results are also very close. 16% of the male students (30 of 185) gave scalar criteria, compared to 17% of the female students (10 of 58). Male and female students were essentially identical in this respect.

4.3.3 Emotional responses to the concepts

We noted several criteria that expressed emotional responses to the concepts on the cards. These included "words I hate" (NS06), "things that cause me grief" (IS03), "things I'm comfortable with" (BS10), "comfortableness" (NS06), "how comfortable I am on the topic" (QS08), "overall likeness of what I do" (MS05)1, and "usefulness to me"(MS05). Surprisingly, five of these responses came from five different male subjects, and two came from a single female subject (MS05). It is surprising that more of these criteria were provided by male subjects than female, but as with the creative analogy criteria, the numbers are so small that any conclusion is tentative. In fact, again as with analogies, the low frequency of these criteria is itself striking. It may indicate that students do not organize their programming knowledge on this basis; alternatively it is possible that subjects believed these criteria to be insufficiently "serious," and therefore did not mention them to the researchers.

	16-24	*25-40*	*over 40*	*Total*
Number of students	187	41	15	243
Men	147	28	10	185
Women	40	13	5	58
Number of sorts	867	170	52	1089
Number of categories	3499	727	189	4415
Percent women	21.4	31.7	33.3	23.9
Average sorts / student	4.6	4.1	3.5	4.5
Average categories / sort	4.0	4.3	3.6	4.1

Figure 10: Comparison by age groups.

	16-24			*25-40*			*41+*			*All ages*		
	M	*F*	*all*	*M*	*F*	*all*	*M*	*F*	*all*	*M*	*F*	*all*
Number of students	147	40	187	28	13	41	10	5	15	185	58	243
Number of sorts	676	191	867	115	55	170	40	12	52	831	258	1089
No. of categories	2669	830	3499	473	254	727	143	47	189	3284	1131	4415
Ave. sorts/subject	4.6	4.8	4.6	4.1	4.2	4.1	4.0	2.4	3.5	4.5	4.4	4.5
Ave. categories/sort	4.0	4.4	4.0	4.1	4.6	4.3	3.6	3.9	3.6	4.0	4.4	4.1

Figure 11: Data for gender and age combined.

4.3.4 Unnamed criteria

Finally, we consider the unnamed criteria. Unnamed criteria are those that were not provided a name, for example, “Group 1,” “Forgot to do it,” or “no name.” In general, these were categories where, even after intervention, the subject was unable to give a name to the category. Of the sorts given by female students, 4.68% were unnamed, while 6.24% of the sorts given by male students were unnamed. We do not yet have the relative percentages by student. This is a slight difference, but it seems to confirm our preconception that female students are more verbal.

In summary, we found surprisingly little difference between the male and female students. A more extensive qualitative analysis might reveal differences between the genders. Alternatively, it has been conjectured that computer-science students are a “different breed” that are much more like each other than the larger population of students who have other majors.

4.4 Age

Do students of different ages organize computer concepts differently? In order to answer this question, we divided students into three groups: 16-24 (traditional college-student age, or close to it), 25-40, and over 40. 187 of our students were in or close to the traditional student age, from 16-24. 41 students were in the age-range 25-40, and 15 of them were over 40.

A breakdown of several factors by age is given in Figure 10. The average number of sorts declines with increasing age. The average number of categories per sort increases slightly from the youngest age group to the middle age group, and then drops substantially from the middle age group to the oldest age group (though the number of students in the oldest age group is quite small).

4.5 Age and gender in combination

Computations about gender and age are combined in Figure 11. The average number of sorts is similar for both genders in each age group (except for the over-40 category). In both genders, the average number of sorts declines with increasing age. Unlike the number of sorts, the number of categories per sort is consistently different between men and women. In each one of the age categories, men have fewer categories per sort.

	Students	*Educators*
Number of subjects	243	33
Total number of sorts	1089	171
Total number of categories	4415	638
Average number of sorts per subject	4.5	5.2
Average number of categories per sort	4.0	3.7

Figure 12: Number of sorts, and number of categories per sort, for students and educators.

4.6 Students vs. educators

We also considered the average number of sorts, and the average number of categories per sort, for the educators and the students in our sample. These computations are summarized in Figure 12. Educators, on average, had fewer sorts than students. Since the educators are generally older than the students, this supports the suggestion based on Figure 11 that the number of sorts decreases with age.

The average number of categories per sort does not seem to decrease with age, however, and it is lower for educators than for students. Educators had slightly fewer categories per sort on the average (3.7) than did students (4.0). 52% of the educators did binary sorts. Interestingly, male educators were consistent with students. 40% of male educators did one or more binary sorts. 88% of female educators did one or more binary sorts, but this percentage is based on a small total sample size of female educators (n=8).

4.7 "Don't Know" categories

4.7.1 Process

Rugg and McGeorge ([11]) advise that respondents be instructed "they can use the categories of 'other', 'not sure' and 'not applicable': this identifies areas where a category is being pushed beyond its range of convenience, areas where respondents' knowledge ends, and various other very useful things." All interviews began with verbal instructions that state "You are welcome to use any criteria you like, and any groups you like including 'don't know', 'not sure' and not applicable'."

Since the "don't know" category can reveal useful information about the subjects' knowledge structures, it is important to examine the cards placed in this category carefully. What are the programming terms that students are most frequently placing into the "don't know" category? Do these terms have common characteristics? Given the variation in institutions, experimenters, and countries, we first did a gist analysis of "don't know" among all of the sorts for all subjects. This yielded a variety of phrases including, "things I didn't understand", undefinable", "haven't learned", "unfamiliar", and "in my bad graces".

Since there were students who classified cards in "don't know" in one sort and in a named category in another, it was clear that "don't know" sometimes meant, "don't know in this context". There were also students who always placed the same cards in a "don't know" category for every sort, indicating that the students did not know what the term meant in any context. In the balance of this analysis, we treat these two meanings of "don't know" as the same. 36% of the sorts contained a "don't know" category and 63% of subjects used a "don't know" category at least once.

Observations during the interviews suggested that terms categorized as "don't know" were frequently terms the researchers considered to be more abstract. Our conjecture is that this observation is consistent with the data from our entire population. To determine abstract terms for our analysis, eleven of the researchers were asked to classify each of the stimuli terms as "Concrete" or "Abstract". Figure 13 shows the results of these classifications.

Figure 14 shows the total number of times each concept was placed in a "don't know" category by one of the student subjects.

4.7.2 Discussion

Terms classified as "don't know" are frequently abstract terms. The six stimuli that researchers most frequently placed in the abstract category comprised 53% of the student "don't know" cate-

Number of researchers (of 11) identifying term as abstract	*Stimuli*
11	dependency, decomposition, abstraction, encapsulation
7	state, tree
3-5	object, scope, list recursion, choice, thread, event
1-2	methods, procedure, type, expression, iteration, array, function
0	if-then-else, boolean, parameter, variable, constant, loop

Figure 13: Abstract stimuli, as classified by researchers.

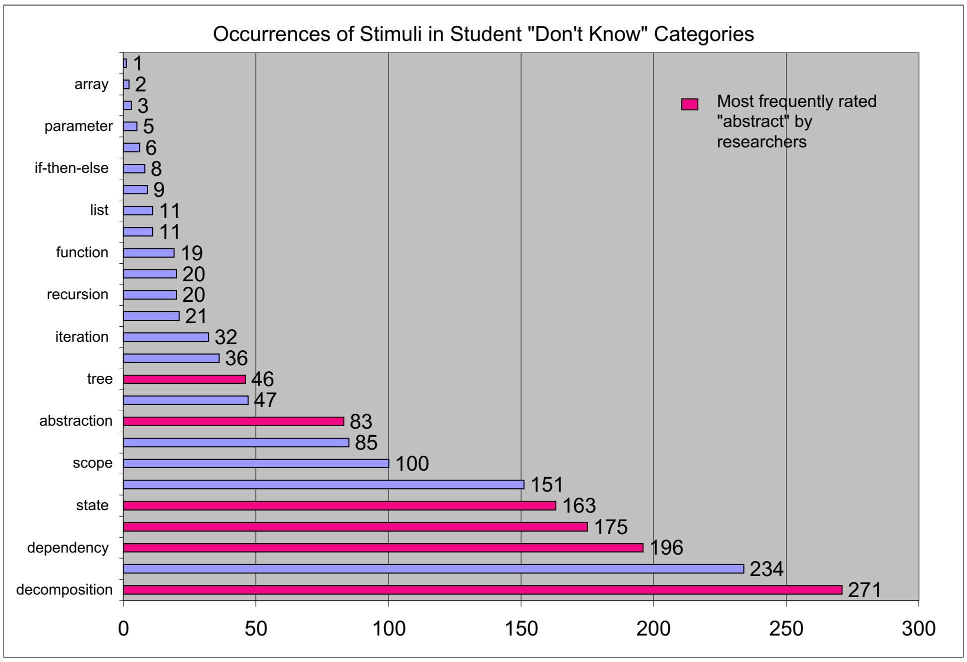

Figure 14: Occurrences of stimuli in student "don't know" categories.

gorizations. If we include "thread," that total would be substantially larger, since it was the second most frequently occurring student "don't know" stimulus. Almost half of the researchers considered it to be abstract as well. Further, 88% of the "don't know" categories contained at least one abstract term. On the other hand, the six cards that none of the researchers categorized as abstract comprised only 1.8% of the student "don't know" categorizations.

4.8 Programming languages used by students

Subjects reported familiarity with a large set of languages (68 in total). Prior to undertaking the card sorting exercise each subject was asked to rate his or her familiarity with Java, C++, C, Ada, Scheme, Pascal, and Visual Basic. In addition, subjects were invited to enumerate "other" languages they were familiar with. No constraints were imposed on subjects' views of what constituted a "programming language". A total of 61 different "other" languages were recorded.

In order, the six most popular languages were C++, Java, Visual Basic, C, Pascal, and Scheme. As shown in Figure 16, 79% of subjects reported some familiarity with C++, and 12.8% had used Scheme. None of the remaining 62 languages had more than 16 mentions (out of 243 students). About 81% of the languages had ten mentions or fewer, and approximately 67% of the languages were "single mentions" by individual subjects.

For each of the seven project-specified languages, plus any others they listed, students indicated their familiarity using an integer score in the range 1-5, with 1 indicating "never used" and 5

	2				*3*				*4*				*5*			
	E		*S*		*E*		*S*		*E*		*S*		*E*		*S*	
	M	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*
C++	5	0	33	10	4	1	36	14	3	0	35	11	3	1	44	9
Java	0	2	18	9	4	2	53	13	6	2	44	10	11	2	21	6
VB	4	4	37	7	7	1	31	16	5	0	21	5	1	1	13	1
C	3	0	44	11	2	5	25	4	6	1	16	2	14	0	8	1
Pascal	3	0	17	6	5	3	10	2	6	1	11	3	11	2	5	3
Scheme	7	1	8	3	5	2	8	4	3	0	6	1	0	0	0	1

Figure 15: Summary of the data regarding language familiarity for the six most frequently reported languages. ("E" stands for "educator," and "S" stands for "student." "1" indicates "never used," and "5" indicates "have used a lot.")

	2		*3*		*4*		*5*		*Any of 2-5*		
	M	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*All*
C++	17.8	17.2	19.5	24.1	18.9	19.0	23.8	15.5	80	75.9	79.0
Java	9.7	15.5	28.6	22.4	23.8	17.2	11.4	10.3	73.5	65.5	71.6
VB	20	12.1	16.8	27.6	11.4	8.6	7.0	1.7	55.1	50	53.9
C	23.8	19.0	13.5	6.9	8.6	3.4	4.3	1.7	50.3	31	45.7
Pascal	9.2	10.3	5.4	3.4	5.9	5.2	2.7	5.2	23.2	24.1	23.5
Scheme	4.3	5.2	4.3	6.9	3.2	1.7	0	1.7	11.9	15.5	12.8

Figure 16: Percentage of male and female students familiar with each of the six most popular languages, with levels of familiarity.

indicating "have used a lot." The familiarity rankings for the six most frequently-cited languages are summarized in Figure 15. C++ and Pascal familiarity is evenly spread across the rating levels while all the others diminish at the higher end.

4.8.1 Programming language familiarity by gender

Using Figure 15 as a starting point, we computed the percentage of men and women who are familiar with each of the top six programming languages. The results of this computation are presented in Figure 16. The top six languages are the same for both men and women, and they appear in the same order. Men appear to rate themselves higher; this suggests a question for further investigation, whether men and women of similar ability generally give themselves different ratings on this kind of scale.

Another question, one that we might be able to answer from our data, is whether men and women report knowledge (at least at some level) of the same number of languages, on average.

4.8.2 Programming language familiarity for educators vs. students

Similarly, we can use the data from Figure 15 as a basis for comparing the students and educators among our subjects. The percentages of students and educators who report familiarity with the six most popular languages at various levels is given in Figure 17. As we might expect, the percentage of educators is generally higher than the percentage of students who report knowledge of a given language. The percentage of educators who report knowledge of C, Pascal, or Scheme is much greater than the percentage of students who report knowledge of those languages. The one exception, surprisingly, is C++. 79% of students report some knowledge of C++, while only 51.5% of the educators report knowledge of C++.

The frequency of these languages is also different in the two groups. Students, as mentioned above, report knowledge of C++ most frequently, followed by Java, VB, C, Pascal, and Scheme in that order. For educators, however, C and Pascal are the most frequent (tied at 93.9%), followed by Java (at 87.9%), then with a significant dropoff to VB, Scheme, and C++, in that order. The

	2		*3*		*4*		*5*		*Any of 2-5*	
	E	*S*	*E*	*S*	*E*	*S*	*E*	*S*	*E*	*S*
C++	15.1	17.7	15.1	20.6	9.1	18.9	12.1	21.8	51.5	79.0
Java	6.1	11.1	18.2	27.2	24.2	22.2	39.4	11.1	87.9	71.6
VB	24.2	18.1	24.2	19.3	15.1	10.7	6.1	5.8	69.7	53.9
C	9.1	22.6	21.2	11.9	21.2	7.4	42.4	3.7	93.9	45.7
Pascal	9.1	9.5	24.2	4.9	21.2	5.8	39.4	3.3	93.9	23.5
Scheme	24.2	4.5	21.2	4.9	9.1	2.9	0	0.4	54.5	12.8

Figure 17: Percentages of students and educators familiar with the six most popular languages, with levels of familiarity.

frequency of C and Pascal is not surprising, given their popularity until recently, but again, the result for C++ seems anomalous. Possibly some of the students are learning C++ in high school, which would explain the fact that their frequency is higher than the frequency of educators at the same institutions. But that still does not explain why relatively few educators report knowledge of C++.

4.8.3 Most frequently occurring card sorts

To determine whether knowledge of particular languages has an effect on category formation, we counted how often each category was formed, in other words, the number of sorts where that combination of stimuli occurred as the *entire* contents of a category. Then for each of the six languages that were most popular with our subjects, we counted the number of times each category appeared in the sorts performed by students who reported some knowledge of that language.

The top ten categories overall were:

1. List, Tree, Array (occurred in 104 out of 1247 sorts)
2. Thread (52 times)
3. Recursion, Loop, Iteration (48 times)
4. Function, Method, Procedure (38 times)
5. If-Then-Else, Recursion, Loop, Iteration (33 times)
6. Decomposition, Abstraction, Encapsulation (28 times)
7. List, Array (28 times)
8. Thread, Event (28 times)
9. List, Tree (27 times)
10. Object, List, Tree, Array (24 times)

The category appearing most frequently across the entire population, "list", "tree", and "array", was language-independent - it was the most frequent category in each of the six language subpopulations by a large margin. The top ten categories for the six language subpopulations all included at least six of the ten most frequent categories from the overall list, and categories 1-3, 6, and 8 from the overall list – "list, tree, array," "thread," "recursion, loop, iteration," "decomposition, abstraction, encapsulation," and "thread, event" – appeared on all of the language subpopulation lists. These similarities indicate that at least some aspects of subjects' conceptual structures are consistent across language.

4.8.4 Functional-first students

Only 20 students assigned themselves a familiarity score of three or higher for a functional language, Scheme, and 17 of those came from a single functional-first institution. This makes it difficult to distinguish effects of language choice from other institutional influences. The functional-first students had characteristic "don't know" categories, and members were likely to be most similar overall to another functional-first student. They grouped "recursion" and "iteration" together in categories more often than the overall population, but not by a large margin. (It is also worth noting that "iteration" occurs more frequently with "recursion" across all subjects than does any other term, supporting previous results with novice computing students.

Most educators expressed familiarity with Scheme, Lisp, or Haskell, but this was not true of the 243 students. None knew Haskell and only two expressed any familiarity with Lisp, neither of which rated themselves above a two. Twenty student subjects assigned themselves a familiarity score of three or higher for a functional language, and in all 20 cases the language was Scheme. Seventeen of these students came from a single institution, making it difficult to distinguish effects of language choice from other institutional influences.

Previous work with novice students has shown that imperative programmers form strong associations between recursion and iteration, since they scaffold their concept of recursion off of iteration. [9] This result is evident in our results as well. Across all 1258 sorts, "recursion" and "iteration" appeared in the same category 896 times. No other card appeared with "recursion" as often. The next closest, "if-then-else", appeared 634 times. We found that students introduced to programming via functional languages associate the terms more closely than the student population average, but the group score is not unique. The 17 students from the functional-first institution grouped the terms together in 85 of their 110 sorts. Thus, "recursion" and "iteration" appeared in the same category in 77% of the sorts done by functional-first students versus 66% for the overall population. While no institutional average over 77% has yet been found, at least one institution had an average of 76%.

A more significant difference was that the functional-first group members had a distinctive pattern to their "don't know" categories, which often contained terms like "thread", "event", "encapsulation", and "dependency".

Gist analysis revealed another distinctive characteristic that is likely the result of learning to program in a language without destructive modification: Ten of the 17 students performed sorts on a criterion having to do with mutability (e.g. "Changeability", "Amount affectable", "Variability", "Things that change and things that stay the same"). In the remainder of the student population, only six sorts made this same distinction.

Analysis of the data relating to programming languages suggests several additional questions, including:

- whether procedural-first and object-first students have different associations with "Object." Conjecture: it's paired with "List" in a procedural-first group, while Object-First Java students associate it with "Variable."
- whether functional-first students think differently about the notion of "state". Here we might look for patterns in the dendrograms.
- whether there are interesting differences between students who have used multiple languages and those who have used just one?
- whether there are interesting differences between students who have been taught the same language with different approaches? (For example, Java taught procedural-first vs. objects-first.)

5 Conclusions and Future Work

In this study, we used a multiple, subject-defined, single-criterion card sort to elicit students' conceptual structures. Unlike observational or programming task studies, card sorts allow us to elicit the conceptual structures formed by students. Gathering data across multiple schools and nations provides insight on structures common across a wide variety of students as well as structures particular to subgroups of students.

Our initial analysis has several results:

- The concepts students place in a "don't know" category are most often abstract.
- Women consistently have more categories per sort.
- The average number of sorts decreases with age.
- The most frequently formed category (list, tree, array) is independent of the languages with which a student is familiar.
- It is suggested that concepts learned early are easy, and concepts learned late are hard.
- Students have a strong association between recursion and iteration.
- Functional-first students appear to have distinctive "don't-know" categories.
- Functional-first students appear to be more likely to perform sorts based on "mutability" or "change."

Some of these results need to be qualified. For example, there is strong support for the proposition that the more abstract a concept is, the more likely students are to place it in a "don't know" category. Not all students distinguished clearly between two types of "don't know" categories, however, so sometimes these categories include concepts the student doesn't know, and sometimes they are concepts the student knows, but can't fit into any of the other categories.

When examining the possible influence of programming language on concept formation, we considered the most popular categories. The most frequently formed category overall – list, tree, array – it turns out, is also most frequent among all of the subpopulations of students who know C++, or Java, or one of the other popular languages among our subjects. Several of the top-ten categories overall are also top ten regardless of language. We have not yet looked at the top-ten categories in any other context, so we don't know if there are subpopulations where this varies: male/female, old/young, by institution, etc. In addition, when we looked at programming languages, we grouped students by whether they reported having any exposure to a language at all. How much influence would a minimal exposure to a language have on concept formation? We would like to examine this question further, taking into account the students' CS1 language, or the language they report knowing the best.

Another very interesting result is really only suggested by our data: the idea that concepts learned early are easy, and concepts learned late are hard. Only eighteen students did these sorts, and it's not clear how many of them did both (easy-hard and early-late). The correlation so far is based on aggregate data. Those who did do both sorts, if any, may all have learned things in the same order; they may even have been from the same institution. To be sure of this result, we'd need to know what order students learned things in and include students who learned things in different orders. In particular, it would be good to have some students who learned abstract concepts such as "encapsulation" early, because it may be that the real correlation here is that concrete things (which happen to have been learned early) are easy, and abstract things (which happen to have been learned late) are hard.

Finally, the number of functional-first students was also very small, and nearly all of them were from a single institution. Thus, the results concerning functional-first students, while interesting, must also be considered no more than suggestive.

In addition to the results of the study we also have some meta-results, things we learned about the process of carrying out a study on this scale.

- Although we used the McCracken task as the participation discriminator, it required a subjective estimation by individual researchers and may have yielded students at different levels of preparedness. Similarly, the necessities of scheduling the sorting interviews meant that students who sorted later may have had more computer science instruction.
- This study used educators but did not concentrate on collecting a wide cross-section of experts. We have little information about the educators, such as their level of expertise or whether they have taught the students in this study.

- The meaning of "don't know" is problematic. Some researchers had their students make explicit which "don't know" they meant, but others did not.
- Conclusions based on subject self-rating of programming language experience reflect the confidence of the individual and may be skewed along the lines of race, gender or ethnicity. Although we attempted to standardize evaluations of student performance, ratings for students may not be uniform across institutions. In addition, this evaluation of student performance may not take into account factors such as the first programming language used or the amount and source of previous programming experience (e.g. AP credit, transfer credit).
- Data analysis of a sufficiently rich and interesting data set always takes longer than you think.

The data set gathered in this study represents the largest and most diverse collection of material relating to conceptual structures of first competency programmers to date. We have identified several themes in the data, but there is much more to be done; this rich data set will support further investigation.

Moreover, if our data is combined with new material, more wide-ranging questions might be addressed. For example, an in-depth study of educators and industry professionals may provide insight into their conceptual structures, which might be related to their background, training, and career, which may provide a richer comparator for student hierarchies. A longitudinal study tracking student conceptual structures may provide insights into how these change over time and how they correlate with an expert population.

Acknowledgements

The Bootstrapping project was supported by the National Science Foundation Grant No. DUE-0122560. We are grateful to Barbados Community College for data gathering opportunities; to Gordon Rugg, for advice and guidance on card sorting; to Robin Blume-Kohout for advice on matrix comparability; to Ian Utting for propping up the workshop leaders; to Kate Deibel and Janet Davis for help with data collection; to Karen Furuya for help with data analysis; and to Noel Welsh and Jim Bender.

Authors' Affiliations

Marian Petre Computing Department, The Open University, Milton Keynes, MK7 6AA, UK. m.petre@open.ac.uk

Sally Fincher Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK. S.A.Fincher@kent.ac.uk

Josh Tenenberg Computing and Software Systems, University of Washington Tacoma, Tacoma, WA 98402, USA. jtenenbg@u.washington.edu

Ruth Anderson Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA. ruth@cs.virginia.edu

Dennis Bouvier Computer Science Department, Saint Louis University, St. Louis, MO 63103, USA. bouvier@cs.slu.edu

Sue Fitzgerald Metropolitan State University, Minneapolis, MN 55403, USA. sue.fitzgerald@metrostate.edu

Alicia Gutschow Information Systems Technology, Blue Ridge Community College, Weyers Cave, VA 24486, USA. gutschowA@brcc.edu

Susan Haller Computer Science Department, University of Wisconsin Parkside, Kenosha, Wisconsin 53141, USA. haller@cs.uwp.edu

Gary Lewandowski Math/CS Department, Xavier University, Cincinnati, OH 45207, USA. lewandow@cs.xu.edu

Raymond Lister Software Engineering Department, University of Technology, Sydney, Australia. raymond@it.uts.edu.au

Renée McCauley Computer Science Department, College of Charleston, Charleston, SC 29424, USA. mccauley@cs.cofc.edu

John McTaggart Department of Mathematics and Computer Science, Drake University, Des Moines, IA 50311, USA. john.mctaggart@drake.edu

Briana B. Morrison Computer Science Department, Southern Polytechnic State University, Marietta, GA 30060, USA. bmorriso@spsu.edu

Laurie Murphy Computer Science and Computer Engineering Department, Pacific Lutheran University, Tacoma, WA 98447, USA. murphylc@plu.edu

Christine Prasad School of Computing and Information Technology, UNITEC Institute of Technology, Auckland, New Zealand. cprasad@unitec.ac.nz

Brad Richards Computer Science Department, Vassar College, Poughkeepsie, NY 12604, USA. richards@cs.vassar.edu

Kate Sanders Math/CS Department, Rhode Island College, Providence, RI 02906, USA. ksanders@ric.edu

Terry Scott Department of Mathematical Sciences, University of Northern Colorado, Greeley, CO 80639, USA. tscott@fisher.unco.edu

Dermot Shinners-Kennedy Department of Computer Science and Information Systems, University of Limerick, Limerick, Ireland. Dermot.Shinners-Kennedy@ul.ie

Lynda Thomas Department of Computer Science, University of Wales, Aberystwyth, Dyfed, SY23 3DB, Wales, UK. ltt@aber.ac.uk

Suzanne Westbrook Computer Science Department, University of Arizona, Tucson, AZ 85721, USA. sw@cs.arizona.edu

Carol Zander Computing & Software Systems, University of Washington Bothell, Bothell, WA 98011, USA. zander@u.washington.edu

References

[1] Beth Adelson. Problem solving and the development of abstract categories in programming languages. *Memory and Cognition*, 9(4):422–433, 1981.

[2] M. S. Aldenderfer and R. K. Blashfield. *Cluster Analysis*. Sage Publications, 1984.

[3] C. M. Allwood. Novices on the computer: A review of the literature. *International Journal of Man-Machine Studies*, 25:633–658, 1986.

[4] M. T. H. Chi, R. Glazer, and M. J. Farr, editors. *The nature of expertise*. Erlbaum, 1988.

[5] Simon P. Davies, David J. Gilmore, and Thomas R. G. Green. Are objects that important? the effects of expertise and familiarity on the classification of object-oriented code. *Human-Computer Interaction*, 10(2 & 3):227–248, 1995.

[6] Jianming Dong, Shirley Martin, and Paul Waldo. A user input and analysis tool for information architecture. (http://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/410), June 2004.

[7] Michael McCracken et al. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4):125–140, December 2001.

[8] M. W. Eysenck and M. T. Keane. *A Handbook of Cognitive Psychology*. Psychology Press, 1995.

[9] C. M. Kessler and J. R. Anderson. Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2, 1986.

[10] F. Marton and R. Säljö. On qualitative differences in learning: Outcome and process. *British Journal of Educational Psychology*, 46:4–11, 1976.

[11] G. Rugg and P. McGeorge. The sorting techniques: Card sorts, picture sorts and item sorts. *Expert Systems*, 14(2):80–93, 1997.

Appendices

A Background Questionnaire

Subject Identifier: ____________________
Age: ____________
Gender: M F
Are you a: full-time part-time student?
Do you generally attend day night classes?
Do you hold an external job? Yes No
If so, how many hours per week (on average) do you work? ____________
If so, is the job in the computer industry? Yes No
What programming language was taught in your (CS1) class? __________
What programming language was taught in your (CS2) class? __________
At what age did you begin to program? ____________

A.1 Programming Experience

On a scale of 1 (never used) to 5 (have used a lot) please rate your familiarity with the following programming languages (place an X in the appropriate column):

	1	2	3	4	5
Java					
C++					
C					
Ada					
Scheme					
Pascal					
Visual Basic (VB)					
Other (please list)					

Thank you.

B Instructions for Card Sort

B.1 Introduction

You will be given some cards to sort. Each card will have the name of a programming concept on it. We would like you to sort the cards into groups, using one criterion at a time. When you have finished sorting, we will ask you what the groups were that you sorted the cards into, and what the criterion was for that sort. Once this has been done, we would like you to sort the cards again-using a different criterion-and then to keep on sorting them until you have run out of criteria.

For example, if the task was sorting pictures of different types of house, you might sort them into groups "brick", "stone", "wood", etc., depending on their main material of construction; the second time you might divide the cards into groups called "one", "two" and "three", depending on the number of floors in each building.

In this task, we would like you to concentrate on how programs are constructed, rather than on superficial surface detail. For instance, if you were sorting pictures of houses, you might sort the houses in a variety of ways relating to construction, such as whether they required deep foundations, or whether the brickwork would be complicated, or whether there were internal load-bearing walls, rather than on superficial details such as the colour of the brick.

You are welcome to use any criteria you like, and any groups you like, including "don't know", "not sure" and "not applicable". The main thing is to use only one criterion in each sort-please don't lump two or more in together. If you're not sure about something, just ask.

You may have noticed that the cards are numbered: this is for convenience when recording the results. The numbering is random, so please don't use that as a criterion for sorting!

If you have any comments or questions, then please say, and we will sort them out.

Thank you for your help.

C History of the Bootstrapping Project

Bootstrapping Research in Computer Science Education was a project intended as a hands-on "way in" to high-quality computer science education research for computer science higher-education faculty. The project was supported by the National Science Foundation Grant No. DUE-0122560 and by Washington State's Institute of Technology at the University of Washington-Tacoma. Bootstrapping used a workshop format to bring practitioners and expert researchers together to initiate principled, large-scale teaching and learning research.

The key objectives of the project were:

- To improve the state of computer science education research and thereby to improve the state of CS education by developing skills (in the design, conduct, and management of research) of CS educators by exposing them to relevant theory and methods.
- To establish research relationships that extend beyond the duration of the workshops, contributing to a research community able to sustain a constructive discourse and ongoing collaboration.
- To engender skills and confidence that allow participants to initiate subsequent research and engage in the wider research community.

C.1 Participants

The workshop leaders chose twenty-one workshop participants from over sixty applicants. The participants came from a range of host institutions including community colleges, primarily undergraduate colleges, and research universities from four continents and six countries. The participants brought diverse expertise, enthusiasm, and a high-level of energy to the project.

C.2 First workshop

Participation in the project included a commitment to attend two workshops given one year apart and to perform the study described in the experiment kit in the interim. The initial workshop took place June 1 - June 6, 2002 in Port Townsend, Washington. This workshop provided an "entry point" into theoretical and empirical perspectives on Computer Science Education Research. The workshop took a "trading zone" approach by borrowing heavily from other disciplines including education, adult learning, business, engineering, HCI and artificial intelligence, cognitive psychology, and other social sciences. The workshop emphasized the "Six Guiding Principles of Scientific Research" detailed in the National Research Council Report Scientific Research in Education. Participants used the workshop to build collaborative relationships with other CS higher-ed faculty.

Sessions on research methodology covered the design of research projects, including the importance of focal questions, evidence and analysis. The framing of research questions and how to investigate them were key topics. Other topics included research ethics, approaches to annotated bibliographies, and record keeping.

To complement the lecture and discussion, the participants practiced elicitation techniques such as structured and unstructured interviews and laddering. Additionally, the participants learned to do card sorts, the elicitation technique used in the experiment kit. The experiment kit introduced in this project addresses programming understanding and conceptual foundations of programming skills in first- and second-year undergraduate students.

C.3 Intervening Activities

During the following year, all workshop participants conducted card sort interviews of students and educators within their own institutions. Participants maintained contact with one another through the project mailing list and via informal meetings during the SIGCSE Technical Symposium. Additional activities carried out included their institutional requirements for human subjects research. Participants also performed preliminary analysis on their own subject data.

C.4 Capstone Workshop

Twenty of the twenty-one participants returned for the June 7 - June 12, 2003, capstone workshop with data in hand. Over 270 subjects and experts had been interviewed. Data was combined and anecdotal results were shared. The hard work began with the analysis of the data. Every participant was responsible for formulating questions about the data and analysing. This project paper was written by all participants, and plans were made to report and disseminate results more widely to the Computer Science Education community, through workshops, conferences, journal publications etc.

A critical focus of the second workshop was that each participant prepared a research plan for a further study, either individual or in collaboration, related to their own interests. These were critiqued and revised in groups and in plenary. In addition, plans for future collaborations and further analysis of the project data were discussed.