Strategies that Students Use to Trace Code:
An Analysis Based in Grounded Theory

Sue Fitzgerald
Information and Computer Sciences
Metropolitan State University
St. Paul, MN 55106 USA
+1 (651) 793-1473

sue.fitzgerald@metrostate.edu

Beth Simon
Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114
+1 (858) 534-5175

esimon@cs.ucsd.edu

Lynda Thomas
Department of Computer Science
University of Wales
Aberystwyth, Wales UK SY23 1BJ
+44 (1970) 622452

ltt@aber.ac.uk

ABSTRACT

How do first year students approach problems which require them to read and understand code? We report on a grounded theory-based analysis of student transcripts from 12 institutions where students were asked to "think aloud" when solving such problems. We identify 19 strategies used by students. Primary results are that all students employ a range of strategies, there were, in total, many different strategies that were applied, students use multiple strategies on each individual problem, students applied different strategies to different types of questions, and students often applied strategies poorly. We show that strategies conform with existing education theories including Bloom's Taxonomy and the Approaches to Study Inventory. Additionally, we discuss emergent theories developed through a card sort process.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer Science Education.

General Terms

Theory.

Keywords

Grounded Theory, Problem Solving, Strategies, Multiple Choice Questions (MCQs), Multi-institutional, Card Sort, Tracing, Test Taking Strategies, First Year Students.

1. INTRODUCTION

In this paper, we report on an analysis of students' ability to understand code. This work began during the ITiCSE 2004 working group which investigated student reading and tracing skills by evaluating student success at completing 12 multiple choice questions (MCQs) featuring arrays and loops [11]. The working group, itself a follow-up of McCracken's 2001 ITiCSE

working group that reported on first-year students' inability to program [12], found that many students have fragile knowledge which inhibits their ability to systematically and manually exercise a piece of code. Here, we focus on a subset of the Lister *et al* corpus of data to investigate what strategies students used in answering code-based MCQs.

This is an exploratory study, utilizing the set of 37 transcripts that were collected by the members of the ITiCSE 2004 working group. The majority of the work presented here focuses on two representative questions (Q2 and Q8) from the set of 12. The exact text of Q2 and Q8 is shown in Figure 1.

The primary motivation of this work seeks to understand how students approach problems that involve understanding and tracing code. We elicit such information (recorded in transcript form) using a "think aloud" format designed to get students to answer questions of the form "What are you thinking?" and "Why did you do that?"

The goal is to identify strategies used by students from analysis of the transcripts and to examine these strategies and their use to develop theories of how students approach the reading and understanding of code.

This emergence of theories from the data is part of a *grounded theory* approach that originates in the work of Glaser and Strauss [6]. Glaser and Strauss outlined grounded theory as a way to "[discover] theory from data systematically obtained from social research" (p. 2). Theories emerge organically from the data through a process of collection, coding and analysis of data. These activities should "be done together as much as possible. They should blur and intertwine continually, from the beginning of an investigation to its end" (p. 43).

In this paper we describe the process by which we utilized "think aloud" transcripts to identify strategies used by first year students in answering code-based MCQs. We define and provide a narrative of examples of these strategies from transcript data. Then we describe a number of theories that arise from these strategies that describe broader classifications and explanations of the strategies that student use. It is this qualitative identification of strategies – and the theories that we suggest arise from it – that is the fruit of this grounded theory-based research, rather than a quantified coding of exactly how often these strategies are found in student work. Although traditional grounded theory research claims that analysis of data should be intertwined with data collection in order to inform data collection procedures, that was not possible in the structure of this working group study.

Question 2.	Question 8.

Consider the following code fragment.
 	
``java
int[] x1 = {1, 2, 4, 7};
int[] x2 = {1, 2, 5, 7};
int i1 = x1.length-1;
int i2 = x2.length-1;
int count = 0;
while ((i1 > 0) && (i2 > 0)) {
    if (x1[i1] == x2[i2]) {
        ++count;
        --i1;
        --i2;
    }
    else if (x1[i1] < x2[i2]) {
        --i2;
    } else { // x1[i1] > x2[i2]
        --i1;
    }
}
``
 	
After the above while loop finishes, "count" contains what value?
 	
a) 3
b) 2
c) 1
d) 0
	If any two numbers in an array of integers, not necessarily consecutive numbers in the array, are out of order (i.e. the number that occurs first in the array is larger than the number that occurs second), then that is called an <u>inversion</u>. For example, consider an array "x" that contains the following six numbers:
	
	4 5 6 2 1 3
	
	There are 10 inversions in that array, as:
	
	x[0]=4 > x[3]=2
	x[0]=4 > x[4]=1
	x[0]=4 > x[5]=3
	x[1]=5 > x[3]=2
	x[1]=5 > x[4]=1
	x[1]=5 > x[5]=3
	x[2]=6 > x[3]=2
	x[2]=6 > x[4]=1
	x[2]=6 > x[5]=3
	x[3]=2 > x[4]=1
	
	The skeleton code below is intended to count the number of inversions in an array "x":
	
	``java
	int inversionCount = 0;
	
	for (int i=0; i<x.length-1; i++) {
	   for (xxxxxx) {
	      if (x[i] > x[j])
	          ++inversionCount;
	   }
	}
	``
	
	When the above code finishes, the variable "inversionCount" is intended to contain the number of inversions in array "x". Therefore, the "xxxxxx" in the above code should be replaced by:
	
	a) for(int j=0; j<x.length; j++)
	b) for(int j=0; j<x.length-1; j++)
	c) for(int j=i+1; j<x.length; j++)
	d) for(int j=i+1; j<x.length-1;j++)

Figure 1: Multiple Choice Questions Analyzed.

Q2 requires selection of a variable value, Q8 requires the reader to fill in the correct line of missing code

Section 2 provides a discussion of the context and methodology of the experiment and more details on grounded theory and Section 3 discusses work related in both content and method. The strategies that emerged in this study are shown in Table 1 and are discussed in Section 4. In Section 5 we look at a number of theories that are supported or suggested by the data and in Section 6 discuss future work. Section 7 states our conclusions.

2. CONTEXT OF THE EXPERIMENT AND METHODOLGY

In this section we describe the context in which this data was gathered as part of an ITICSE 2004 working group study on the reading and tracing of code. Additionally, we describe the methodology followed in the collection and analysis of the data.

2.1 The ITICSE Working Group Study

The ITICSE 2004 working group on reading and tracing code led by Raymond Lister consisted of 12 participants from 8 countries. Most students who undertook this performance test were "first competency" programming students [12]. Most had either completed, or had nearly completed, their first semester of studying programming at their respective institutions. In some institutions, the MCQs were used as part of the procedure for assigning a final grade to the students. In other institutions, students volunteered to be part of the study. More details can be found in the working group report [11].

2.1.1 Performance Data

Each working group member tested students on the 12 MCQs, under exam conditions. The primary data collected was the students' answers for each MCQ, from which a score out of 12 could be calculated. A total of 941 students contributed data to this part of the study, but of those students only 556 students were given all twelve questions (some students took part in the test at earlier stages).

We selected Questions 2 and 8 for further study because they were representative of the skills examined in the test. Question 2 involved understanding and tracing a piece of code and was answered correctly by 65% of the students. Question 8 involved filling in one line of missing code to perform a task and was answered correctly by 51% of students. The overall average score on the entire set of 12 questions was 60% and the median score was 66.7% (N=556).

2.1.2 Written Data

Students were given "scratch" paper upon which they were allowed to draw pictures or perform calculations as part of answering the MCQs. [11] provides a discussion of the relationship between these annotations and performance.

2.1.3 Interview Transcripts

The 12 working group members interviewed a total of 37 students. In the interviews, students were asked to "think out loud" as they answered the core set of MCQs. The interviews were recorded and then transcribed. Some were translated into English at transcription time.

Some differences in data collection occurred including the exact timing of the think aloud (during an intital testing or after a completed test) and also in instructor prompting and response to student think aloud activities (ranging from no instructor input to detailed instructor questions at points). However, all of the data was deemed to be of comparable quality and, for the purposes of this study, minor differences in collection method appear to be neutralized by the quantity of data collected.

2.2 Methodology and Definitions

While general problem-solving strategy work has a significant literature (see Section 3), in this paper we specifically define a **strategy** as an approach used by a student to get to an answer to one of the MCQ tracing-type questions.

Tracing, as used in this paper, refers to the overall process of trying to emulate, at some level of detail and/or accuracy the process of a computer executing code. Tracing is an entire process that may or may not involve **doodling** – some form of physical annotation on a piece of paper. In contrast, we'll later describe a strategy that strikes close to the heart of the tracing process called walkthroughs. A **walkthrough** is a verbal strategy identified in the transcripts where students talk through some level of code emulation. Students performing walkthroughs did not always doodle and vice versa.

In the development and analysis of strategies used by students, we followed a grounded theory-based approach to develop theory emergent from the think aloud transcript data. Initially strategies were identified by a preliminary reading of all 12 questions of the 37 transcripts. Strategies were then further refined by three independent researchers' readings of all 37 student transcripts of Q2 and Q8. In these readings the researchers underlined and noted student phrases and words that indicated a strategy was being employed. Final strategies (and their names) were firmly identified via a group review proccess. This process resulted in the identification of 19 strategies that students used to get to an answer in code-based questions (based on Q2 and Q8). Next, the researchers individually returned to the transcripts and coded which strategies were used by each student on Q2 and Q8. This led to codings related to each of the questions. The codings are not the focus of this report. Informally, we found significant inter-rater reliability of this codings, but we only report on them in the context of individual students using the same strategies – sometimes successfully and other times not.

After completing this process we observed that most students used multiple strategies in working on a given question, that different strategies were used for different questions, and that many strategies that were used successfully by certain students would be used incorrectly or unsuccessfully by other students (see Section 5.1). In order to uncover the theories underlying student use of strategies in code-based problem solving, we performed unrestricted card sorts [15] of the strategies identified. These sorts led to the recognition of existing theories and development of emerging theories that are discussed in Sections 5.2 and 5.3.

3. Related Work

Understanding Code. Educators have long understood the difficulties which students experience in learning to program [16].

An excellent recent overview of research in this area can be found in [14].

A careful consideration of the problem makes it clear that this is not actually very surprising. As du Boulay pointed out, the skill of learning to program a computer has many facets and beginners are faced with them all at the same time, including not only the syntax of the language but also the notional machine on which programs are run [3].

More recently, McCracken *et al* assessed the programming ability of a large population of students from several universities, in the United States and other countries [12]. The authors tested first year students on a common set of programming problems. The majority of students performed much more poorly than expected. In fact, most students did not even get close to finishing the set task. Numerous reasons were cited including: having no idea how to solve the exercise, not enough time, inadequate designs and inability to implement designs.

Lister *et al* sought to clarify the results of McCracken's work [11]. Lister's question was "to what degree do [programming students perform poorly] because of poor problem solving skills, or because of fragile knowledge and skills?" In other words is the problem a design problem or a language problem? If we further restrict our consideration to the underlying skills necessary for understanding code, we note that only 10% of college students may be found to correctly answer questions in propositional logic [7], a serious handicap in understanding the kind of code on which our MCQs were based.

Multiple Choice Questions. In the study of test taking in general, MCQs have been the subject of much study. In the realm of Computer Science [10] discusses use of MCQs to get at higher level assessment as defined by Bloom's taxonomy. In [9] Lister specifically claims that CS1 students should be examined via multiple choice in order to fairly assess students. Additional work has been done on permutational MCQs which are purported to address issues of guessing, trivial recognition of facts, and construction versus choice [5]. A brief discussion of a set of other optional marking schemes for MCQs (especially those administered electronically) can be found in [2].

Grounded Theory. Grounded Theory is an "emergent methodology" originally proposed by sociologists Glaser and Strauss in 1967 [6] – though a number of variants have been developed since. Several web resources for getting a basic background in grounded theory exist [17] [18].

4. Strategies

A review of the transcripts revealed a striking breadth and variety of strategies exhibited by the subjects as they sought answers to the test questions. An initial examination of the transcripts revealed 19 distinct strategies. Given the limited experience of these students, most of them in their first computer programming class, the sheer variety of techniques used by students in attempting to understand the questions and to come up with solutions was of interest.

Table 1 identifies the strategies identified from the transcripts of Q2 and Q8. Further explanation of each strategic approach is given next, along with examples taken from student transcripts.

Reading the question

The transcripts revealed that some subjects explicitly and carefully read through the problem statement before attempting to solve the problem. They previewed the question and made sure they understood what was being asked. For the more difficult questions, those involving selection of a code segment rather than determination of the value of a variable, reading the question often involved understanding the meaning of the problem or discerning the intent of the code by reading the textual explanation. For the simpler problems, involving finding the value of a variable, reading the question might mean determining in advance which variable's value was sought.

Some transcript data that identified students using a "reading the question" strategy include:

"If any two numbers in an array of integers, not necessarily consecutive numbers in the array, are out of order, that is - the number that occurs first in the array is larger than the number that occurs second - then that is called an inversion." [Taken literally from the problem statement for Question 8, A031, A038]

"This time I'm going to look to see what the question wants first. So trying to find the value of count." [Q2, A037]

In contrast, some subjects deliberately chose to read code before studying the question.

"Basically, I'm reading the code to understand what's going on, then I'll read the question." [Q2, Q002]

Previewing the code by identifying data structures

Another common beginning strategy was the identification of program components on the syntactic level, specifically by noting the data structures. Subjects employing this strategy were not attempting to explain the meaning of the data or control structures. Rather, they were simply recognizing and naming the parts of the program.

"You've got two integer arrays" [Q2, L005]

"x1 is an array with four integers, x2 is an array with four integers." [Q2, S001]

Previewing the code by identifying the initialization of data structures

Closely connected to previewing the code by identifying data structures was the strategy of recognizing the initialization of variables or data structures.

"int array x1 is equal to 1, 2, 4, 7. int array 2 is equal to 1, 2, 5, 7." [Q2, N003]

"There are 2 variables i1 and i2 that start at 3." [Q2, L019]

Although the identification of data structures and the initialization of data structures would seem to naturally be related, they were not always used together. In some cases initialization was noted, but the type of data structure was not identified.

"…the initial values of I1 and I2 were both 3" [Q2, C002]

Previewing the code by identifying control structures

Similar to the identification of data structures is the identification of control structures. Some subjects chose to locate loop statements, nested loops and if statements as a way of

Table 1: Strategies that students used as identified on Q2 and Q8.

Strategy	Explanation	Example
Reading the question	Previewing the question, looking for what is asked	<<see Section 4>>
Previewing the code by identifying data structures	Identifying program components on the syntactic level; an explanation of meaning is not mentioned	That's an array, a is an int
Previewing the code by identifying the initialization of data structures	Identifying program components on the syntactic level; an explanation of meaning is not mentioned	i = 0
Previewing the code by identifying control structures	Identifying program components on the syntactic level; an explanation of meaning is not mentioned	There's a nested loop
Understanding new concepts: Semantic	Understanding something new, a new concept, by relating prior understood knowledge to less understood knowledge or by using an example	inversion – so, 4, 5, 6 there are 10 inversions here
Pattern Recognition: Temporally self-referential	Syntactic; recognizing that this looks like something from another question on this test; multiple choice distractors	This goes to x.length-1 like problem #1
Pattern Recognition: Outside knowledge	Syntactic; recognizing from outside knowledge; something I'm familiar with (but not on this test)	Loops always go from 0 to length-1
Pattern Recognition: Seeking higher levels of meaning from the code	What the code really does at a higher semantic level	This is like a sort; it selects the even numbers
Walkthroughs	Tracing, testing boundary or error conditions	so the condition is while i1>0 which it is
Strategizing	How would I write the code, what would I need to do?	I'm trying to figure out how I would do this …
Grouping	Looking for similarities and differences in the answers; selecting more than one answer at once, possibly for elimination or inclusion	Answers A and D are alike
Differentiation	Noticing differences between answers or lines of code or choices	Answer A is i < j and Answer B is i <= j
Elimination	Eliminating specific choices	It can't be A or D
Guessing	Explicitly stated	I guessed. I just picked one.
Thoroughness	After selecting an answer, checking for correctness of **other** solutions just to be sure	I'm pretty sure that C works but I'll just check that D doesn't
Starting over	Getting lost, recognizing an error, simply starting again	I'm backtracking. I'm starting over
Coming back to the question later	Going on to another part of the test without completing this problem	I'll come back to this later.
Posing questions	Explicitly questioning	What is the value of i here? What is the program doing here? What do they want?
Doodling	Annotations on the test paper	see [11] for details

understanding the problem. The detection and recognition of control structures usually occurred as the subject began to look at the problem, but occasionally it occurred later in the solution process.

"Uh, Oh, so I just noticed that it's a nested loop." [Q8, P002]

"it wants to loop until and including the last element" [Q8, Q003]

It is interesting to note that when using this approach most subjects focused exclusively on looping constructs; if statements were more typically scrutinized while doing walkthroughs. This may be an artifact of the type of problem used; looping drives the problem. Alternatively, it may be a recognition on the part of the subjects that looping behavior is inherently more difficult to understand.

Understanding new concepts: Semantic

This category is applies to semantics (meaning) rather than syntax (form). Subjects sometimes identified an idea with which they were not familiar. In those cases, many subjects constructed an understanding of this new idea by relating it to some previously understood information, or by using an example to make understanding clearer. This strategy demonstrated an ability to extrapolate from existing knowledge.

This approach is most frequently seen in Q8 where the notion of an inversion is introduced. Subjects were not intended to have previously been exposed to the concept of inversion, and an explanation and examples were given. For most subjects, this question demanded new understanding on the semantic level.

"I'm reading over the question. Finding out what an inversion means or is. So I see that it's when a number occurs first in the array is larger than the number that occurs second. And it just gives an example so you understand." [Q8, A037]

"This one took me a while because I couldn't figure out what inversion was, and then I looked at it again and said OK well this is bigger than that. .. Because this is bigger than that and so on and so forth. And it took me a couple of minutes to figure what inversion was." [Q8, C001]

Pattern Recognition: Temporally self-referential

Pattern recognition is a powerful learning and test-taking tool. It takes several forms. In some cases subjects recognized that one test question resembled another test question and made assumptions based on previous answers. Unfortunately, these assumptions often led to the wrong answers – the multiple choice question distractors.

"Number 2 says, consider the following code fragment. Again, basically I did the same thing again as the first one." [Q2, O002]

"And the second problem I did is, it's like what I did before, but I just look at these two, so I can just guess, like what it's saying is like when the numbers are different?" [Q2, O003]

Pattern Recognition: Outside knowledge

Subjects also recognized patterns learned in other settings. This category, limited to syntactic or data-structure level semantic pattern recognition, is distinct from the following category, "Pattern Recognition: Seeking higher levels of meaning from the

code". Subjects often used this pattern recognition to understand looping behavior.

"Actually I think this one was not that hard. OK, so it's going to go from 0 to length - 1, which means the whole interval, I mean the whole array." [Q8, O002]

"First of all I knew that in the for loop you're going to compare j to length-1, because the way the arrays are 0 through the number of elements minus 1." [Q8, O001]

Pattern Recognition: Seeking higher levels of meaning from the code

In addition to the mundane use of pattern recognition to identify the components of a program, pattern recognition was also used to intuitively understand what the code was doing at a higher semantic level. Here subjects related the test question to previously understood algorithms, such as sorts.

"OK so it's like a comparison." [Q2, P002]

"So once again this is just a fancy method of sorting I would think" [Q8, Q002]

"Um, yeah, I knew that it would count the number of equal elements." [Q2, Q003]

Walkthroughs

When using the most frequent strategy, code walkthroughs, subjects traced verbally through the code logic and/or updated the values of variables while working through the sample code. Walkthroughs were used to predict the values of variables, test boundary or error conditions, and test hypothesized results. This strategy was often, but not always, associated with written annotations or doodles [11]. Although walkthroughs were indeed the most common strategy employed, not all subjects used it. This strategy includes partial walkthroughs, not necessarily thorough, which were often employed.

"int x1 is equal to 1, 2, 4, 7 and int x2 is equals to 1, 2
while i1 > 0
i1, i1 is x1.length which is x1.length = 4
i1 so i1 = x1.length-1
x.length is 4-1
is 4 – 1
just 3 and i2, x2.length which is 4-1 which is 3
i2 = 4-1 which is 3
so both of these are true so I enter the loop" [Q2, P001]

Strategizing

The test questions were loosely grouped into two categories – those that asked the subjects to predict the results of a code segment (e.g.,, Q2) and those that asked the subjects to select the correct set of missing lines of code (e.g., Q8). When faced with the problem of choosing a matching answer for the missing code, some subjects asked themselves how *they* would write the code or what they would need to do in order to solve the problem. That is, instead of trying out the answers given to them, these subjects asked themselves what code they would need to write or what they would need to do to solve the problem.

"And it's just sort of … what an inversion is and what this code is trying to do now just sort of thinking what I would logically put in there before I look at any of the answers" [Q8, N001]

"Trying to work out what sort of code would follow.…int j cannot be... am trying to, you know, trying to find out which, which would be the missing code. So for that I am trying to sort of, think out and work out how the code would sort of look like, you know the structure of that code fragment." [Q8, N003]

"Now I'm writing some code, then I'll try to check the number of inversion in array x" [Q8, E002]

Grouping

A number of strategies directly related to taking multiple choice tests emerged. Subjects often engaged in grouping or looking for similarities and differences in the answers or selecting more than one answer at once, possibly for elimination or inclusion. Grouping strategies were often paired with the differentiation or elimination strategies described below.

"So I narrowed it down to two answers. and then it left me choosing whether to initialize j to 0 or i+1" [Q8, O001]

"Yes, so that one has the same problem as the first one … C has the same problem as A and I can tell it's pretty much comparing … there are only two differences really. A and C have the same problems and B and D have the same problems." [Q8, N002]

Differentiation

Another strategy related directly to taking multiple choice tests, differentiation involved noticing differences between answers or lines of code.

"So the problem is, is it x.length or x.length-1?" [Q8, P001]

"So I narrowed it down to two answers. and then it left me choosing whether to initialize j to 0 or i+1, and I chose i+1 because what you're counting in the loop is dependent upon i, so I don't know whether that's a good explanation, but that's how I did it." [Q8, O001]

Elimination

Elimination, often used with grouping and differentiation, involves the removal of some MCQ choices.

"So it can't be A. So now I'm trying. But I'm not going to bother with B because it's starting out at the beginning of the array for j too. So now I'm looking at C. … so it would skip the last value in the array so D would not work so the answer is C." [Q8, A037]

"So can't be first two. j=i+1 I see we have to start off at i+1 and go towards the ... you have a choice between going towards the actual number or ... look at this example, if try to go all way to position 6 we'll have an array index out of bounds error so want it to go to 6-1 cos that'll be the actual array index – so it must be 4" [Q8, L005]

Guessing

Perhaps the most amusing strategy is guessing. Subjects employ guessing when they have no idea what the correct answer is (pure guessing) and sometimes after reducing the number of viable choices through elimination (educated guessing).

"I was totally lost on this one, I just took a wild guess." [Q8, H002]

"There was like 5 minutes left and I just picked what I thought looked best at that time." [Q8, H003]

"It would be a complete guess to be honest." [Q2, L002]

Thoroughness

A small number of subjects checked their work after identifying the correct answer by checking all the remaining alternatives.

"So it seems that one would work. I'm going to go ahead and try D real fast." [Q8, A37]

Starting over

Some subjects recognized that they had become confused and needed to begin again.

"Now I'm looking at the question again and wondering, I'm just going to start over because I didn't write down enough." [Q8, A37]

"Once again I'm going to do the same thing … I messed everything up" [Q2, Q002]

Coming back to the question later

A related strategy was to leave the question and return to it later.

"I'll skip that one and go back to it." [Q8, P001]

Posing questions

When puzzling out the answers to a question some students asked questions. This strategy may simply be a stylistic artifact. However, the transcripts indicate deeper thinking in some cases.

"Ten, why are there ten?" [Q8, T001]

"Can this be an endless loop?" [Q2, T003]

"What am I looking for?" [Q2, P003]

Doodling

We refer the interested reader to the discussion on doodling or written annotations given in [Lister].

5. Theories

Analysis and immersion in the transcript data produced the list of utilized strategies presented in Section 4. This was a first step in the development of theory grounded in empirical data. According to Glaser and Strauss, some theories should become clear from examination of the data. When we came to analyze our data, we found some theories emerged in this fashion (organically and obviously).[2] These are theories that are directly evident from the data collected and are reported on in Section 5.1. Some theories were more indirectly spurred by consideration and discussion of the data. We propose these theories in Sections 5.2 and 5.3 – but note that full elicitation of these theories from the data would best

---

[2] Only strategies relevant to theories are listed in tables in this section.

Table 2: Sample students and the strategies they employed

ID	Question 2 strategies	Correct	ID	Question 8 strategies	Correct
L005	Reading the question, Previewing the code by identifying data structures, Previewing the code by identifying the initialization of data structures, Walkthrough	yes	T001	Reading the question, Understanding new concepts: Semantic, Strategizing Differentiation, Elimination, Walkthrough	yes
N003	Reading the question, Previewing the code by identifying data structures, Previewing the code by identifying the initialization of data structures, Previewing the code by identifying control structures, Walkthrough	no	N003	Reading the question, Understanding new concepts: Semantic, Strategizing, Grouping, Elimination	no

be guided by further, more specialized, data collection. This is in line with grounded theory practice where data collection should be informed by theory development (not something immediately possible in the working group format).

5.1 Evident Theories

Several things were immediately obvious from an examination of the identified strategies and where they were observed.

1. All students employed a range of strategies.
2. Many strategies were applied overall.
3. Students use multiple strategies on each individual problem.
4. Students applied different strategies to different questions.
5. Students often used good strategies poorly.

These observations can be made throughout our coding of the transcripts but for a clear example see Table 2. Recall that Q2 and Q8 were selected as representatives of the two types of problems on the test: obtaining output from code, and determining which of a set of options should be used to fill in a line of code.

Students L005, T001 and N003 used a wide range of strategies (observation 1). There were many strategies that were applied overall (observation 2), and students used multiple strategies on each individual problem (observation 3). The two problems elicited different strategies (observation 4). But although students used similar, possibly "good", strategies, L005's application of them resulted in success on question 2 and N003 in failure on the same question (observation 5). Likewise on question 8, T001's application of strategies resulted in success and N003's application of similar strategies, in failure. More discussion of student performance and strategies can be found in Section 5.4.

5.1.1 Computing versus Test Taking Strategies

To discuss another immediately evident theory, we need to return to the original problem and area of research interest. We were interested in whether a 'language problem' existed – that is, could students read and trace code. We were not primarily focused on the fact that these issues were examined in the light of multiple choice questions.

In examination of the data set it was obvious that some strategies seem more crucial for our central research question than others. In particular, Pattern Recognition: Temporally self-referential, Grouping, Differentiation, Elimination, Guessing and Coming back to the question later all seem to be strategies related to test-taking rather than understanding code. This leaves us then with the following pool of 12 strategies:

- Reading the question
- Previewing the code by identifying data structures
- Previewing the code by identifying the initialization of data structures
- Previewing the code by identifying control structures
- Understanding new concepts: Semantic
- Pattern Recognition: Outside knowledge
- Pattern Recognition: Seeking higher levels of meaning from the code
- Walkthroughs
- Strategizing
- Thoroughness
- Starting over
- Posing questions

Parts of the list are still problematic. For instance, some of the remaining strategies might be test-taking or might be code related (Thoroughness and Starting over, for example). Others need further investigation. Is 'posing questions' merely a rhetorical device? Or does it elicit other issues related specifically to code understanding? These issues might merit further study, but we can report with certainty that some strategies are more related to test taking (and some specifically to MCQ-style test taking) and others more directly to code reading and tracing.

5.2 Conformance of the Strategies to Existing Theories

There is some evidence that the strategies outlined above conform with existing theories about learning and problem solving. Clearly, more work in this area is needed, but we present preliminary analysis based on the available data.

Table 3: Strategies as Organized by Structure within Bloom's Taxonomy

Bloom	Keywords	Strategies
Knowledge	Recognition	Pattern Recognition: Outside knowledge
Comprehension	Understanding	Previewing the code by identifying data structures Previewing the code by identifying the initialization of data structures Previewing the code by identifying control structures
Application	Using concepts	Walkthrough
Analysis	Breakdown of material and relationships	Understanding new concepts: Semantic
Synthesis	Skill in writing / creation	Strategizing
Evaluation	High level comparison of alternatives	Pattern Recognition: Seeking higher levels of meaning from the code

Table 4: Strategies as Organized by Deep versus Surface versus Strategic Learning

Approach	Characteristics	Strategies
Deep	Intends to understand learning material Looks for patterns and underlying principles Relates ideas to previous knowledge	Pattern Recognition: Seeking higher levels of meaning from the code Understanding new concepts: Semantic Strategizing Walkthrough?
Surface	Intends to fulfill the immediate task Attempts to memorize facts Finds it difficult to make sense of new ideas Does not search for patterns or connections	Reading the code Previewing the code by identifying data structures Previewing the code by identifying the initialization of data structures Previewing the code by identifying control structures Pattern Recognition: Outside knowledge Walkthrough?
Strategic	Intends to do well in assessments Focuses on assessment criteria	Test-taking strategies fit here

5.2.1 Bloom's Taxonomy and Strategies

One popular educational framework is Bloom's Taxonomy of Educational Objectives where learning skills are divided into six graduated levels. These are, from lowest to highest: knowledge, comprehension, application, analysis, synthesis and evaluation [1]. Strategies utilized by the students also clearly fell into similar levels of cognitive complexity as shown in Table 3.

5.2.2 Deep, Surface and Strategic Approaches to Learning

Another categorization of learning strategies is found in the work of Entwhistle and others and has resulted in the Approaches to Study Inventory [4]. For instance Richardson [13] describes how students "manifest a number of different approaches …that are dependent on the context, the content, and the demands of the learning tasks." The approaches may be categorized as shown in Table 4 where we matched appropriate strategies with each approach.

In particular, the Walkthrough strategy was interesting. Lister was disappointed [8] that so many students solved questions like Q2 by 'cranking through the code'. He would see this strategy as a surface approach, but since the questions had little deeper meaning, following the logic might *be* the deeper meaning for beginners.

5.3 Emergent Theories

Although it was possible to fit the strategies that were observed into existing theories of learning, other theories, evidenced by different ways of categorizing strategies, are also suggested by the data.

In order to find ways of categorizing the strategies, each of the three researchers performed unconstrained card sorts [15] on the strategies. Although the naming of the resulting categories differed, many of the actual groupings showed remarkable similarity. These led to a consideration of emergent theories, two of which are presented below.

5.3.1 Temporal Groupings

As described in our methodology, 19 strategies were identified and, later, coded. Before the second more detailed examination of the data, these strategies were in fact grouped into several main sets. These sets correspond, more or less, with a 'temporal grouping' which later emerged in the card sorts. The groups are shown in Table 5. These temporal groupings reflect observed

Table 5: Temporal Grouping

Time	Group	Strategy
Early ⬇ Late	Initialization / Familiarization	Reading the question Previewing the code by identifying data structures Previewing the code by identifying the initialization of data structures Previewing the code by identifying control structures Understanding new concepts: Semantic
	Recognition	Pattern Recognition: Outside knowledge Pattern Recognition: Seeking higher levels of meaning from the code
	Modeling	Walkthrough Strategizing
Outside time		Thoroughness Starting over
Final step	Selection (MCQ specific)	Grouping, Differentiation, Elimination, Guessing

Table 6: Syntax, Semantics and Pragmatics

Programming Language Construct	Strategy
Syntax related strategies	Reading the question Previewing the code by identifying data structures Previewing the code by identifying the initialization of data structures Previewing the code by identifying control structures Pattern Recognition: Outside knowledge
Semantics related strategies	Understanding new concepts: Semantic Walkthrough
Pragmatics related strategies	Pattern Recognition: Seeking higher levels of meaning from the code Strategizing

time-based and process-based behaviors that were some of the most evident within the transcripts.

5.3.2 Syntax, Semantics, and Pragmatics

An evaluation of programming languages is often conducted in terms of syntax or grammar, semantics or how it works, and pragmatics or how it can be used ('you use a loop when you want to go through all the elements of a collection').

The student strategies can also be grouped in this way (see Table 6). While we might hope that students follow all of these strategies in appropriate situations, we might expect that more successful students would use sets of strategies that span all of these categories. Particularly, students who only use syntactic strategies are unlikely to solve dynamic tracing problems. Similarly, students who fail to grasp the semantics of the problem are not likely to correctly choose the missing lines of code.

5.4 Special Consideration: Success and Strategies

An initial desire in conducting this research was that we would be able to determine what strategies are likely to lead to success versus those that are likely to lead to failure. For example, the ITiCSE researchers hypothesized that 'doodles' which showed evidence of tracing the code would be correlated with success and, in fact, that does seem to be borne out by the data [11].

In our examination of strategies, again, no such clear correlations are available. For example, we thought that 'Walkthough' would be an "obvious" successful strategy, but almost all students used it – some successfully, some not. Success was determinant not just on *which* strategies were employed but rather, the crucial factor emerged in "how well" the particular strategy was employed. When we examine in more detail, almost all students did walkthroughs on Q2 but many were not complete, were not careful, or had semantic difficulties that led to crucial mistakes. In this case particularly, in order to better develop theories on strategies that lead to

success, a more refined data collection approach needs to be devised.

6. Future Work
In keeping with the grounded theory framework, this work should serve to inform further studies where proposed theories can be used to develop more specialized data collection. For example, time augmented transcripts could be utilized to identify temporal strategy use. Different questions could be developed to more specifically target the levels of Bloom's taxonomy and tested to see if certain strategies are more correlated with particular questions.

Specifically, a more detailed and structured think aloud might need to be developed in order to define theory to explain why use of similar strategies can lead to such different results. Why and where do students "go wrong" in applying a strategy? Were they attempting to emulate a strategy demonstrated by an instructor in a classroom setting – but perhaps had fragile or incomplete understanding of that strategy? Were they attempting to use a strategy that they use in solving homework-style problems where a computer is available for experimentation? Can students recognize for themselves when their strategies are going wrong?

Finally, would the same strategies (at least those identified as more code-related) occur in a non-multiple choice environment? Would students continue to use such a broad variety of strategies if they had an open-form response?

7. Conclusions
In this work we report on the strategies employed by first year students in seeking to read and understand code. Primary findings show that many different strategies are used by students and that all students employ a range of strategies – even on individual problems. Additionally, problem structure affected the type of strategies used and, finally, students often employed strategies poorly – perhaps indicating fragile knowledge of how to read and trace code.

This data is based in a grounded theory-based analysis of transcripts from 37 students (from 12 institutions) performing a think aloud when answering two different questions requiring the ability to read and understand code. This study was part of a larger ITiCSE Working Group project led by Raymond Lister in 2004.

Further analysis of and immersion in the original working group data led to development of emergent theories. Further refinement of these theories will evolve through more detailed data collection on student approaches to code-based problem solving.

8. Acknowledgements

9. References
[1] Bloom, B. S., Mesia, B. B. and Krathwohl, D. R. *Taxonomy of Educational Objectives* (two vols: The Affective Domain & The Cognitive Domain). Addison-Wesley, 1956.

[2] Bush, M. Alternative Marking Schemes for On-Line Mulitople-Choice Tests. *In Proceeedings of the 7th Annual Conference on the Teaching of Computing.* 1997.

[3] du Boulay, J. B. H., O'Shea, T. and Monk, J. The black box inside the glass box: Presenting computing concepts to novices. In E. Soloway and J.C. Spohrer (Eds), *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, 1989, 431-446. Reprinted from du Boulay, O'Shea and Monk (1981).

[4] Entwistle, N.J. and Tait, H. *The Revised Approaches to Study Inventory*. Edinburgh: Centre for Learning and Instruction, University of Edinburgh, 1994.

[5] Farthing, D. W., Jones, D.M., McPhee, D. Permutational Multiple-Choice Questions: An Objective and Efficient Alternative to Essay-Type Examination Questions. *In Proceedings of the 3rd Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '98),* (Dublin City University, Ireland, August 18-28, 1998). ACM Press, New York, NY, 1998, 81-85.

[6] Glaser, B.G. and Strauss, A.L. The discovery of grounded theory: Strategies for qualitative research. Aldine: Chicago, 1967.

[7] Lawson, A.E., Karplus, R. and Adi, H. The acquisition of logic and formal operational schemata during the secondary school years. *Journal of Research in Science Teaching, 15*, 6 (1978), 465-478.

[8] Lister, R. Personal communication, 2005.

[9] Lister, R. Objectives and Objective Assessment in CSl. In *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education (SIGCSE 2001)* (Charlotte, NC, USA, February 21-25, 2001). ACM Press, New York, NY, 2001, 292-296.

[10] Lister, R. On Blooming First Year Programming, and its Blooming Assessment. In *Proceedings of the fourth Australasian computing education conference (ACE 2000)* Melbourne, Australia, December, 2000). ACM Press, New York, NY, 2000, 158-162

[11] Lister, R., Adams, E., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. and Thomas, L., A Multi-National Study of Reading and Tracing Skills in Novice Programmers. *ACM SIGCSE Bulletin, 36*, 4 (December 2004), 119-150.

[12] McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I., and Wilusz, T. A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-Year CS Students. *ACM SIGCSE Bulletin, 33*, 4 (2001), 125-140.

[13] Richardson, J. T. E., Using Questionnaires to Evaluate Student Learning: Some Health Warnings. Gibbs, G. (ed.) *Improving Student Learning – Theory and Practice*. Oxford: Oxford Centre for Staff Development, 1994. (http://www.city.londonmet.ac.uk/deliberations/ocsd-pubs/isltp-richardson.html)

[14] Robins, A., Rountree, J., and Rountree, N. Learning and Teaching Programming: A Review and Discussion, *Computer Science Education*,*13,* 2(2003), 137-172.

[15] Rugg, G., and McGeorge, P. The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems, 14*, 2 (1997), 80-93.

[16] Soloway, E. and Iyengar, S. (Eds). Empirical Studies of Programmers. Ablex, NJ, USA, 1986.Soloway, E. and Spohrer, J. (Eds). *Studying the Novice Programmer.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[17] Borgatti, S. Introduction to Grounded Theory, http://www.analytictech.com/mb870/introtoGT.htm (last accessed April 29, 2005).

[18] Grounded theory: a thumbnail sketch, http://www.scu.edu.au/schools/gcm/ar/arp/grounded.html (last accessed April 29, 2005).