

# 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 92093-0404 USA  
+1 (858) 534-5175

esimon@cs.ucsd.edu

Lynda Thomas

Department of Computer Science  
University of Wales  
Aberystwyth, Wales UK SY23 3DB  
+44 (1970) 622452

lth@aber.ac.uk

## ABSTRACT

How do beginning 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, Think Aloud, Test Taking Strategies.

## 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 [15]. The working group, itself a follow-up of McCracken's 2001 ITiCSE

working group that reported on beginning computing students' inability to program [17], found that many students have fragile knowledge which inhibits their ability to systematically and manually exercise a piece of code. Fragile knowledge is defined as when "a student may be able to articulate particular items of knowledge when explicitly prompted for any of them, (but) when that student is asked to apply that knowledge...the student 'sort of knows'" but can't produce a correct answer [18] [17]. 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 [7]. 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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICER'05, October 1–2, 2005, Seattle, Washington, USA.

Copyright 2005 ACM 1-58113-043-4/05/0010...\$5.00.

<p><b>Question 2.</b></p> <p>Consider the following code fragment.</p> <pre> 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 &gt; 0 ) &amp;&amp; (i2 &gt; 0 )) {     if ( x1[i1] == x2[i2] ) {         ++count;         --i1;         --i2;     }     else if (x1[i1] &lt; x2[i2]) {         --i2;     } else { // x1[i1] &gt; x2[i2]         --i1;     } } </pre> <p>After the above while loop finishes, “count” contains what value?</p> <p>a) 3 b) 2 c) 1 d) 0</p>	<p><b>Question 8.</b></p> <p>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:</p> <p style="text-align: center;">4 5 6 2 1 3</p> <p>There are 10 inversions in that array, as:</p> <pre> x[0]=4 &gt; x[3]=2 x[0]=4 &gt; x[4]=1 x[0]=4 &gt; x[5]=3 x[1]=5 &gt; x[3]=2 x[1]=5 &gt; x[4]=1 x[1]=5 &gt; x[5]=3 x[2]=6 &gt; x[3]=2 x[2]=6 &gt; x[4]=1 x[2]=6 &gt; x[5]=3 x[3]=2 &gt; x[4]=1 </pre> <p>The skeleton code below is intended to count the number of inversions in an array “x”:</p> <pre> int inversionCount = 0;  for (int i=0; i&lt;x.length-1; i++) {     for ( xxxxxx ) {         if ( x[i] &gt; x[j] )             ++inversionCount;     } } </pre> <p>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:</p> <p>a) for(int j=0; j&lt;x.length; j++) b) for(int j=0; j&lt;x.length-1; j++) c) for(int j=i+1; j&lt;x.length; j++) d) for(int j=i+1; j&lt;x.length-1;j++)</p>
--	--

**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**

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.

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 METHODOLOGY

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. Researchers were asked to solicit students in their first or second computer science course. In general, students ranged over CS0, CS1, and CS2 courses, though the amount of time students had spent studying at each institution varied widely. 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 [15].

#### 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 encouraged to use their paper tests as "scratch" paper upon which they were allowed to draw pictures or perform calculations as part of answering the MCQs. [15] and [16] provide 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 aloud as they answered the core set of MCQs. The interviews were recorded and then transcribed verbatim by the individual working group members who collected them. Some were translated into English at transcription time.

Some differences in data collection occurred including the exact timing of the think aloud (during an initial 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). Students were selected in different manners at various institutions, with the majority volunteering to participate. Nonetheless, all of the data were deemed to be of comparable quality and, for the purposes of this study, differences 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 process. 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 unconstrained card sorts [21] 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

#### 3.1 Understanding Code

Educators have long understood the difficulties which students experience in learning to program [22] [23]. An excellent recent overview of research in this area can be found in [20].

A careful consideration of the problem makes it clear that this is not 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 [4].

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 [17]. 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 [15]. 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 [9], a serious handicap in understanding the kind of code on which our MCQs were based.

#### 3.2 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, [14] discusses use of MCQs to get at higher level assessment as defined by Bloom's taxonomy. In [13] 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 [6]. A brief discussion of a set of other optional marking schemes for MCQs (especially those administered electronically) can be found in [3].

#### 3.3 Grounded Theory

Grounded Theory is an "emergent methodology" originally proposed by sociologists Glaser and Strauss in 1967 [7] – though a number of variants have been developed since. Several web resources for getting a basic background in Grounded Theory exist [2] [8].

### 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. Quotes are identified by question and subject number in the form [Q2, A037]. The letter in the subject number is an institutional identifier.

#### S1. 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]

#### S2. 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]

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

Code	Strategy	Explanation	Example
S1	Reading the question	Previewing the question, looking for what is asked	<<see Section 4>>
S2	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
S3	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
S4	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
S5	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
S6	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
S7	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
S8	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
S9	Walkthroughs	Tracing, testing boundary or error conditions	so the condition is while i!>0 which it is
S10	Strategizing	How would I write the code, what would I need to do?	I'm trying to figure out how I would do this ...
S11	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
S12	Differentiation	Noticing differences between answers or lines of code or choices	Answer A is $i < j$ and Answer B is $i <= j$
S13	Elimination	Eliminating specific choices	It can't be A or D
S14	Guessing	Explicitly stated	I guessed. I just picked one.
S15	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
S16	Starting over	Getting lost, recognizing an error, simply starting again	I'm backtracking. I'm starting over
S17	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.
S18	Posing questions	Explicitly questioning	What is the value of i here? What is the program doing here? What do they want?
S19	Doodling	Annotations on the test paper	<<see [15] and [16] for details>>

### S3. 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]*

### S4. 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 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.

### S5. Understanding new concepts: Semantic

This category is applied 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]*

### S6. 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]*

### S7. 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]*

### S8. 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]*

### S9. 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 [15]. 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]

### **S10. 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]

### **S11. 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]

### **S12. 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]

### **S13. 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]

### **S14. 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]

### **S15. 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]

### **S16. 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]

### **S17. 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]

### **S18. 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]

### **S19. Doodling**

We refer the interested reader to the discussion on doodling or written annotations given in [15] and [16].

**Table 2: Sample students and the strategies they employed**

ID	Question 2 strategies	Correct	ID	Question 8 strategies	Correct
L005	Reading the question (S1), Previewing the code by identifying data structures (S2), Previewing the code by identifying the initialization of data structures (S3), Walkthrough (S9)	yes	T001	Reading the question (S1), Understanding new concepts: Semantic (S5), Walkthrough (S9), Strategizing (S10), Differentiation (S12), Elimination (S13)	yes
N003	Reading the question (S1), Previewing the code by identifying data structures (S2), Previewing the code by identifying the initialization of data structures (S3), Previewing the code by identifying control structures (S4), Walkthrough (S9)	no	N003	Reading the question (S1), Understanding new concepts: Semantic (S5), Strategizing (S10), Grouping (S11), Elimination (S13)	no

## 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).<sup>2</sup> 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 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 – both by individual students and across various problems.
3. Students used 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.

<sup>2</sup> Only strategies relevant to theories are listed in tables in this section.

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 (S6), Grouping (S11), Differentiation (S12), Elimination (S13), Guessing (S14) and Coming back to the question later (S17) all seem to be strategies related to test-taking rather than understanding code. This leaves us then with the following pool of 12 strategies relevant to Computer Science code reading rather than MCQ style:

- Reading the question (S1)
- Previewing the code by identifying data structures (S2)
- Previewing the code by identifying the initialization of data structures (S3)
- Previewing the code by identifying control structures (S4)
- Understanding new concepts: Semantic (S5)
- Pattern Recognition: Outside knowledge (S7)



- Pattern Recognition: Seeking higher levels of meaning from the code (S8)
- Walkthroughs (S9)
- Strategizing (S10)
- Thoroughness (S15)
- Starting over (S16)
- Posing questions (S18)

Parts of the list are still problematic in terms of the strength of their relevance to Computer Science code reading in contrast to test taking. For instance, some of the remaining strategies might be test-taking or might be code related (Thoroughness (S15) and Starting over (S16), 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 [1]**

Bloom	Keywords	Strategies
Knowledge	Recognition	Pattern Recognition: Outside knowledge (S7)
Comprehension	Understanding	Previewing the code by identifying data structures (S2) Previewing the code by identifying the initialization of data structures (S3) Previewing the code by identifying control structures (S4)
Application	Using concepts	Walkthrough (S9)
Analysis	Breakdown of material and relationships	Understanding new concepts: Semantic (S5)
Synthesis	Skill in writing / creation	Strategizing (S10)
Evaluation	High level comparison of alternatives	Pattern Recognition: Seeking higher levels of meaning from the code (S8)

**Table 4: Strategies as organized by deep versus surface versus strategic learning [5] [19]**

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 (S8) Understanding new concepts: Semantic (S5) Strategizing (S10) Walkthrough? (S9)
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 question (S1) Previewing the code by identifying data structures (S2) Previewing the code by identifying the initialization of data structures (S3) Previewing the code by identifying control structures (S4) Pattern Recognition: Outside knowledge (S7) Walkthrough? (S9)
Strategic	Intends to do well in assessments Focuses on assessment criteria	Test-taking strategies fit here (S10, S11, S12, S13, S14)

### 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 least difficult to most difficult: 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. For example, the Pattern recognition: Outside knowledge (S7) strategy involves the recollection of previous examples. Bloom identifies recognition or recall as belonging to the first or most simple level of his taxonomy of cognitive learning.

Bloom’s second level of abstraction is called comprehension. The strategies of Previewing the code by identifying data structures (S2), Previewing the code by identifying the initialization of data structures (S3) and Previewing the code by identifying control structures (S4) are ways in which students understand the meaning of problems, interpret instructions and state problems in their own words [10].

Walkthroughs (S9) fall into Bloom’s application level. At this level learners apply previous learning in a new context. That is, the skill of tracing code is now being applied to code examples the student has not previously seen.

At the analysis level, Bloom expects to the learner to “see patterns” [11], to “separate material or concepts into component parts so that its organizational structure may be understood” [10]. Another way of putting this is that the learner “distinguishes between facts and inferences” [10]. The strategy of Understanding new concepts: Semantic (S5) fits at this level of Bloom’s taxonomy. Here students identified ideas with which they were unfamiliar and made inferences based on previously understood information or by working an example.

As subjects seek to fill in missing lines of code, some choose to strategize (S10) or ask themselves how they would approach and solve a problem rather than looking first at the answer choices. Bloom calls this process of predicting or drawing conclusions, synthesis.

Bloom’s highest order thinking skill is called evaluation. Here the learner compares and contrasts alternatives, interprets and summarizes. The Pattern recognition: Seeking higher levels of meaning from the code (S8) fits here. Students engage in pattern recognition, but at a high level. They seek meaning from the code by comparing a new problem to known problems such as sorts and counting loops. They seek understand the intent of the problem, as compared to other known problems, such that the answers are meaningful rather than achieved by rote memorization or mechanized tracing.

Although the exact categorization of strategies according to Bloom’s taxonomy may be open to interpretation, it is clear that some strategies engage higher order thinking while others rely more on the lower level skills of memory and recognition.

### 5.2.2 Deep, Surface and Strategic Approaches to Learning

Another categorization of learning strategies is found in the work of Entwistle and others and has resulted in the Approaches to Study Inventory [5]. For instance Richardson [19] 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 (S9) strategy was interesting. Lister was disappointed [12] that so many students solved questions like Q2 by ‘cranking through the code’. He would see this strategy as a surface approach, compared to Pattern Recognition: Seeking higher levels of meaning from the code (S8). However, 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 [21] 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. The first column of Table 5 refers to when, in the course of the transcript, strategies were observed to be used. “Outside time” strategies could be employed at any time and, in a sense, break the timeline of a student approach. “Final step” strategies were used as students were making the final steps of selecting a multiple choice answer. These temporal groupings reflect observed 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 aim 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 [15].

In our examination of strategies we observed no such clear correlations. For example, we thought that Walkthrough (S9) would be an “obvious” successful strategy, but almost all students used it – some successfully, some not. Success seems to be determined not just by *which* strategies were employed but rather by “how well” the particular strategies were 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.

**Table 5: Temporal grouping**

Time	Group	Strategy
Early  Late	Initialization / Familiarization	Reading the question (S1) Previewing the code by identifying data structures (S2) Previewing the code by identifying the initialization of data structures (S3) Previewing the code by identifying control structures (S4) Understanding new concepts: Semantic (S5)
	Recognition	Pattern Recognition: Outside knowledge (S7) Pattern Recognition: Seeking higher levels of meaning from the code (S8)
	Modeling	Walkthrough (S9) Strategizing (S10)
Outside time		Thoroughness (S15) Starting over (S16)
Final step	Selection (MCQ specific)	Grouping (S11), Differentiation (S12) , Elimination (S13), Guessing (S14)

**Table 6: Syntax, semantics and pragmatics**

Programming Language Construct	Strategy
Syntax related strategies	Reading the question (S1) Previewing the code by identifying data structures (S2) Previewing the code by identifying the initialization of data structures (S3) Previewing the code by identifying control structures (S4) Pattern Recognition: Outside knowledge (S7)
Semantics related strategies	Understanding new concepts: Semantic (S5) Walkthrough (S9)
Pragmatics related strategies	Pattern Recognition: Seeking higher levels of meaning from the code (S8) Strategizing (S10)

## 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 beginning computing 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.

This work has value for the computing education community in that it informs us of beginning students’ approaches to reading and understanding code. Transcripts provide clues to what students are thinking, what they did to solve problems and how

well they did it. Linking students' strategies to Bloom's higher-order thinking skills and Entwistle's Study Inventory connect us to bodies of literature and approaches we might not otherwise examine as we seek ways to help our students. Emergent theory related to temporal grouping and programming language constructs (syntax, semantics and pragmatics) provide new avenues to explore as we seek insight into how and where novice programmers go wrong in the problem-solving process.

## 8. Acknowledgements

We gratefully acknowledge Raymond Lister for his always cheerful and brave leadership of the strong-willed and occasionally rebellious 2004 ITiCSE Working Group with additional thanks to the other members of the Working Group for assistance in preparation of this paper. We also wish to express our sincere appreciation to all of the students who so generously gave their time to this project. Finally, we thank Sally Fincher for her helpful discussions on Grounded Theory and to the NSF supported workshops on CS Education Research, led by Sally Fincher, Marian Petre and Josh Tenenber, (DUE-0243242) in which all three authors participated. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## 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] Borgatti, S. Introduction to Grounded Theory, <http://www.analytictech.com/mb870/introtoGT.htm> (last accessed April 29, 2005).
- [3] Bush, M. Alternative Marking Schemes for On-Line Multiple-Choice Tests. In *Proceedings of the Seventh Annual Conference on the Teaching of Computing* (Belfast, Ireland, 1997). CTI Computing, 1999.
- [4] 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).
- [5] Entwistle, N.J. and Tait, H. *The Revised Approaches to Study Inventory*. Edinburgh: Centre for Learning and Instruction, University of Edinburgh, 1994.
- [6] 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 Third 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.
- [7] Glaser, B.G. and Strauss, A.L. *The discovery of Grounded Theory: Strategies for qualitative research*. Aldine: Chicago, 1967.
- [8] Grounded Theory: a thumbnail sketch, <http://www.scu.edu.au/schools/gcm/ar/arp/grounded.html> (last accessed April 29, 2005).
- [9] 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.
- [10] Learning Domains or Bloom's Taxonomy, <http://www.nwlink.com/~donclark/hrd/bloom.html> (last accessed August 18, 2005).
- [11] Learning Skills Program: Bloom's Taxonomy, <http://www.coun.uvic.ca/learn/program/hndouts/bloom.htm>, (last accessed August 18, 2005).
- [12] Lister, R. Personal communication, 2005.
- [13] Lister, R. Objectives and Objective Assessment in CSI. 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.
- [14] 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
- [15] 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.
- [16] McCartney, R., Moström, J.E., Sanders, K., and Seppälä, O. Take note: The effectiveness of novice programmers' annotations on examinations. *Informatics in Education*. 4, 1, (2005), 69-86.
- [17] 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.
- [18] Perkins, D. and Martin, F. Fragile Knowledge and Neglected Strategies in Novice Programmers. In Soloway, E. and Iyengar, S. (Eds). *Empirical Studies of Programmers*. Ablex, NJ, USA, 1986, 213-229.
- [19] 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> )
- [20] Robins, A., Rountree, J., and Rountree, N. Learning and Teaching Programming: A Review and Discussion, *Computer Science Education*, 13, 2 (2003), 137-172.
- [21] 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.

[22] Soloway, E. and Iyengar, S. (Eds). *Empirical Studies of Programmers*. Ablex, NJ, USA, 1986.

[23] Soloway, E. and Spohrer, J. (Eds). *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

---