AN ABSTRACT OF THE THESIS OF

Andrew J. Ko for the degree of Honors Baccalaureate of Science in Computer Science and Psychology presented on May 17, 2002. Title: Individual Differences in Programming, Testing, and Debugging Strategies in a Statistical End-User Programming Environment.

Abstract approved:

_____
Mentor's Name (no titles)

This study intended to investigate two areas of end-user programming: the influence of individual differences on success and whether or not groups of programming, testing, and debugging style would naturally cluster together and provide predictive value. Eighty-six participants, from backgrounds of computer science, psychology, engineering and humanities completed at battery of psychological tests and attempted to complete a timed programming task and testing and debugging task in Stata, a statistical programming environment intended for use by individuals with no programming experience. General intelligence and programming experience were good predictors of programming success. Three types of programming strategies were found: (1) the *programmers* group used their background knowledge to solve the programming task with little effort; (2) the *lost/unmotivated* group tended to exhibit repetitive and shallow problem solving; (3) the *lost/motivated* group tended to search for more information and exhibit more guess and check behavior. There were three types of testing and debugging strategies, but no good predictors of success: (1) the *curious/distracted* group ignored the task and became distracted; (2) the *hesitant/focused* group sought little information and made few attempts to debug; (3) the *active/focused* group sought much information and made many attempts to debug. Future work on the data presented here is proposed.

Individual Differences in Programming, Testing, and Debugging

Strategies in a Statistical End-User Programming Environment

by

Andrew J. Ko

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in

Computer Science and Psychology (Honors Scholar)

Presented May 17, 2002
Commencement June 2002

<u>Honors Baccalaureate of Science in Computer Science and Psychology</u> project of <u>Andrew J. Ko</u> presented on <u>May 17, 2002</u>.

APPROVED:

_____

Co-Mentor, representing Computer Science

_____

Co-Mentor, representing Psychology

_____

Committee member, representing Computer Science

_____

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Andrew J. Ko, Author

ACKNOWLEDGEMENTS

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

LIST OF TABLES (continued)

Individual Differences in Programming, Testing, and Debugging Strategies in a Statistical End-User Programming Environment

Introduction

In the past decades, personal computers have become common and pervasive. Accountants rely on computer software to manage constantly changing tax laws, teachers use computers to help children overcome subtle learning disorders, and the growing presence of the Internet foreshadows dramatic changes in interpersonal communication. Many of computers contributions to productivity have come in the form of *automation.* Yet now, many users are realizing that further productivity gains—whether or not such gains are real—may require further knowledge about computers. The accountants who want to write smarter spreadsheets must learn visual basic. The professors who want to make customized instructional software have to learn some kind of programming language. Each has reached a ceiling: without programming skills, such professionals must rely on the software industry to understand and meet their demands. This requires time and money, not to mention the patience and drive of the professionals to learn new software.

Possibly these so-called end users are actually wasting time and resource by trying to program. In fact, there has been evidence that end-user computing has failed to show the productivity gains that some hoped computers would provide (Alavi, 1985). Further, end users may also be wasting time asking others for help with software that does not increase productivity anyway. These issues are worth investigating further.

Yet an issue looms which poses a larger threat. Numerous studies show that spreadsheets, one of the most used forms of end-user programming, contain numerous faults (logic errors), previously undetected by their users and creators (Panko, 1998). Financial professionals often make important hiring and business decisions with spreadsheets, while teachers often track grades with spreadsheets. Even considering only these isolated cases, the difficulty of programming spreadsheets (and the tendency of end users to do it poorly) has the potential to have great impact on their decisions. Thus not only can the difficulty of programming cause inefficiencies in our society, but also the outcomes and success of our commerce. Quite possibly, end users are experience difficulty in other domains as well, affecting similar outcomes.

This problem poses a number of challenges to researchers. How can we make programming a task that is:

- Easier to learn,

- Less prone to faults, and

- More approachable?

In order to inform our approach to solving these problems, the logical first step is to understand the phenomenon of end-user programming.

Exploratory Research in End-User Programming

It is important to note that much of the exploratory and descriptive research on end users and computers focuses more on general computing rather than programming specifically. Nevertheless, the research is important in framing the context of end-user programming, particularly with regard to characteristics of end users themselves. For

example, user attitudes were reported to be a major factor in the success of MIS (Zmud, 1979); more recent studies by Howard and Smith (1986) and Igbaria and Parasuraman (1989) have found that computer anxiety was closely related to the success of end-user computing in business. Possibly a more anticipated result, Rivard and Huff (1988) found that computer background had significant effects on the success of end-user computing; such results are supported by (Cheney et al., 1986; Delone, 1988; Igbaria et al., 1989), all of which report on the same factors of success in end-user computing. From a personality perspective, Zmud (1979) and Matta & Kern (1991) found that locus of control, ambiguity and extroversion/introversion can predict the success of computer information systems. The research seems to indicate that with regard to general software use in the workplace, psychological and background characteristics of users seem to be the best predictors of successful software use.

One of the more comprehensive descriptions of the problem of end-user programming is Bonnie Nardi's *A Small Matter of Programming*. Nardi argues utilizing users' knowledge of their domain may be the most effective way to facilitate end users. This suggests that new interaction techniques such as visual programming (Chang, 1980) or developing artificially intelligence programming systems such as programming by demonstration (Cypher, 1993) may not be the most effective solutions.

It is obvious even from this brief summary of exploratory research that end-user programming research still lacks a basic understanding of an appropriate way to approach making programming an easier, less error-prone, and more approachable task. Nardi makes a case for domain-specific programming, but there is little evidence to show that Chang and Cypher are wrong about visual programming and programming by

demonstration. Despite the lack of evidence to suggest the efficacy of any particular approach, there have been numerous attempts to test each thoroughly. We will discuss selected examples of such research and assess the impact of each on our understanding of end-user programming.

The Psychology of Programming

Newell and Card argued early on that programming languages often make programming more difficult than necessary by ignoring a variety of human-computer interaction issues (1985). For example, in the C programming language a missing semi-colon can generate a slough of error messages that often have nothing to do with the original error. Such a situation is potentially confusing for an end user trying to learn the language, not to mention distracting for a professional programmer.

Thus one approach to understanding end-user programming has been to research *how* people program. Lewis and Olson (1987) were some of many to describe programming as the process of transforming a mental plan that is in familiar terms into one that is compatible with the computer. This definition was later clarified by Green and Petre (1996), through the concept termed *closeness of mapping*. A number of studies have identified important problems in current programming languages based on this concept. For example, Hoc & Nguyen-Xuan (1990) have shown that bugs and difficulties often arise because the distance between the mental plan and the required computer representation is too large. Another such study found that the looping controls provided by modern programming languages do not match the natural strategies that most individuals used (Bonar & Soloway, 1989).

Green (1977, 1983) has demonstrated the importance of notations, arguing that every notation highlights some kinds of information at the expense of obscuring other kinds. For example, if a language highlights dataflow, it may obscure control flow. Sinha (1992) and Vessey (1992) showed that when seeking information in programming systems, there must be a cognitive fit between the mental representations and the external representations.

A number of other significant findings have scoped the problem of the psychology of programming: Brooks (1977) described program design in terms of mapping the problem domain to the program domain, suggesting the difficulty of dealing with entities in the programming domain that do not have corresponding entities in the problem domain. In terms of understanding the program, Anderson (1984) showed that novices need an environment where it is easy to check a program fragment before programming further.

Though this is a just a sample of the findings in the area of psychology of programming, the findings are prescriptive: programming environments for end users require a close mapping of the problem to the program, of the problem domain to the program domain, and of the notations used to communicate to the computer to the concepts in the end users mind. Even still, there are a number of questions that still have great importance. As most of the studies mentioned involved skilled programmers or programming languages designed for professional programmers, we must ask if and how end users and end-user programming environments differ. Furthermore, much of the research has focused on the programming language alone and has ignored the environmental support for learning about a programming environment through online help systems, as well as support for testing and debugging programs. Finally, these

studies have grouped all users into one category in search for universal maxims. Might individual differences play a role in how end users learn about an environment, what strategies they use to program, test, and debug?

Syntax and Naturalness

Early attempts at actually making programming easier were not driven by the empirical results from the area of psychology of programming, but rather by intuition. Developers of HyperTalk (Goodman, 1987) and Cobol (Sammet, 1981), for example, made the language syntax seem more like natural language. These languages still have many of the problems identified earlier (Thimbleby, Cockburn, & Jones, 1992). Another common phenomena in end-user programming is that end users will use natural language to converse with the computer (Pea, 1986), a phenomenon which Nardi (1993) argues may in fact make it more difficult for the user to understand computer's limitations.

More recent attempts at involving some level of "naturalness" in programming languages have been to study the language that end users do use. For example, Pane, Ratanamahatana, and Myers (2001) investigated the language that end users naturally use to express solutions to programming problems (in this study, the problem of programming the videogame Pac Man), finding that event-based and constraint statements were much more common than imperative statements. Such data is useful in guiding the approach to developing new language syntax. Further study yet is required to reveal whether or not their findings are applicable to all end users, or whether there are types of end users that prefer employ different methods of expression. Furthermore, as

the study focused on a single domain of programming the game Pac Man, it is not known if their findings are applicable to all domains.

Testing and Debugging

There has been little effort to investigate the problem of debugging in programming languages in general, let alone end-user programming environments (Lieberman, 1997), but recent attempts show promise in solving this problem. For example, the Zstep95 system (Ungar, 1997) keeps a complete history of computation so that execution can be run backwards; many data visualization systems (Myers, 1980; Myers, 1988; Rojansky, 1997) have provided professional programmers the ability to view run time data in order to aid in debugging programs. There have also been significant attempts by Rothermel, Burnett, and Cook (Rothermel, 2000) to bring visual testing of spreadsheets to professional programmers, and recently, end users.

Yet there are still important gaps in testing and debugging research, most notably the lack of empirical data on end users and such systems. Do end users test and debug in a natural environment? If so, how do they go about doing it, with what do they succeed and fail? Furthermore, if systems are in place for testing and debugging, are they approachable, intuitive, and effective for end users?

Important Gaps in End-User Programming Research

Despite numerous attempts to create tools and theories to making programming easier, less prone to faults, and more approachable, it is clear that the end-user programming

community is still in a descriptive, exploratory phase of research. We still know little about why programming is difficult, and more importantly, what can be done to ease this difficulty. To summarize the gaps in the descriptive data on the problem of end-user programming,

- There has been little research in how end users go about learning about a new programming environment;

- Little is known about how end users create code in a novel environment;

- Little is known about how end users test and debug existing code;

- It has not been shown that end users and skilled programmers are or are not different in any qualitative ways; and

Though no single study can answer all of these questions, the study presented here attempts to at least explore the questions and provide descriptive data on each of these gaps in current end-user programming research.

- How do individual differences influence programming, testing, and debugging performance?

- Are there distinct strategies that end users use to learn about an environment?

- Are there distinct strategies that end users use to program in an environment?

- Are there distinct strategies that end users use to test and debug in a programming environment?

- How do individual differences influence programming, testing, and debugging strategies?

- Is there cause to believe that skilled programmers and end users use the same programming, testing, and debugging strategies?

By investigating these questions, we can identify key issues in improving end-user programming.

Method

In order to answer the research questions posed above, an exploratory experiment was designed in which participants would be given a typical textual end-user programming environment to solve a program construction problem and a program testing and debugging problem. In other words, all participants were exposed to the same experimental condition, with the only variation among participants' backgrounds and experience.

Participants

Participants were recruited from undergraduate computer science and psychology courses, as well as introductory courses in statistical hypothesis testing. Students were recruited at the beginning of their course lectures. A brief description of the experiment and the benefits of participation were given and a signup sheet was passed around. Participants were offered extra credit in their class and entry into a raffle for a $100 prize in exchange for their participation. As the study spanned two academic quarters, a raffle was held for each quarter to increase the participants' perceived chances of winning the raffle. Eight of the participants, all psychology students, were required by their instructor to participate as part of the course requirements. They did not receive extra credit, but were entered into the raffle.

The original sample consisted of 86 participants, but those who did not comprehend the experimental materials because of poor English skills or misunderstanding were removed from the sample, leaving 75 participants. Background characteristics are listed

in Table 1. Two-sample unpaired t-tests were performed to test for differences in self-reported ability and experience, statistical knowledge, and performance on the vocabulary and cognition scales (α was set to .05). Participants whose native language was not English performed significantly lower than native English speakers (p = .004); females had significantly less self-reported experience and ability with mathematics (p < .05), programming (p < .001), and software (p < .05); finally, computer science majors reported higher ability and greater experience with mathematics (p < .01) and programming (p < .01).

Major	Psychology			Computer Science			Other			Total		
Gender	Male	Female	Total	Male	Female	Total	Male	Female	Total	Male	Female	Total
Count	8	12	20	21	3	24	12	19	31	41	34	75
Vocab27	.59 (.09)	.57 (.10)	.58 (.10)	.62 (.12)	.68 (.21)	.63 (.13)	.60 (.15)	.58 (.11)	.59 (.12)	.61 (.12)	.59 (.12)	.60 (.12)
VCog I	.57 (.09)	.51 (.10)	.53 (.10)	.54 (.15)	.68 (.20)	.55 (.16)	.48 (.16)	.52 (.18)	.50 (.17)	.52 (.15)	.53 (.16)	.53 (.15)
Statistics Test	.25 (.15)	.33 (.21)	.30 (.18)	.36 (.16)	.27 (.15)	.35 (.16)	.34 (.11)	.34 (.17)	.34 (.15)	.33 (.15)	.33 (.18)	.33 (.16)
Age	20.6 (1.3)	22.4 (3.6)	21.7 (3.0)	25.4 (8.2)	26.7 (7.4)	25.6 (8.0)	24.3 (3.9)	26.1 (6.7)	25.4 (5.7)	24.2 (6.5)	24.9 (5.9)	24.5 (6.2)
Sleep	5.88 (1.89)	6.83 (1.27)	6.45 (1.57)	7.00 (1.22)	6.33 (2.08)	6.91 (1.31)	6.75 (1.22)	7.05 (1.75)	6.94 (1.54)	6.71 (1.40)	6.92 (1.58)	6.80 (1.48)
Mathematics Experience	3.91 (1.52)	2.96 (1.43)	3.34 (1.51)	5.55 (.99)	5.25 (.66)	5.51 (.95)	4.96 (2.28)	4.71 (1.39)	4.81 (1.76)	5.05 (1.65)	4.14 (1.60)	4.64 (1.68)
Statistics Software Experience	1.25 (1.34)	1.93 (1.08)	1.66 (1.21)	1.08 (.99)	2.25 (.75)	1.23 (1.02)	2.15 (.98)	1.26 (1.33)	1.60 (1.27)	1.43 (1.14)	1.59 (1.24)	1.50 (1.18)
Programming Experience	.58 (.85)	.31 (.91)	.42 (.88)	4.24 (1.16)	3.39 (.92)	4.13 (1.15)	2.19 (1.89)	.96 (1.09)	1.44 (1.55)	2.97 (1.99)	.91 (1.29)	2.03 (1.97)
Computer Experience	5.44 (1.45)	5.50 (1.11)	5.48 (1.09)	6.12 (.71)	5.83 (1.26)	6.08 (.76)	6.04 (.72)	5.66 (1.21)	5.80 (1.05)	5.96 (.83)	5.62 (1.15)	5.81 (1.00)
Experiment Attitudes	4.72 (.92)	4.52 (.95)	4.60 (.92)	5.36 (.97)	4.14 (.63)	5.24 (.97)	4.88 (.66)	4.83 (1.12)	4.85 (.96)	5.09 (.90)	4.69 (1.02)	4.91 (.97)
Computer Attitudes	4.28 (1.45)	3.88 (1.19)	4.04 (1.28)	6.45 (.46)	6.17 (.63)	6.42 (.48)	5.48 (.96)	4.89 (1.07)	5.11 (1.05)	5.74 (1.20)	4.64 (1.26)	5.24 (1.34)
Statistics Attitudes	2.81 (1.18)	2.77 (1.58)	2.78 (1.40)	3.43 (.93)	3.83 (.88)	3.48 (.91)	3.98 (.88)	3.32 (1.31)	3.57 (1.19)	3.47 (1.02)	3.17 (1.39)	3.33 (1.20)
Mathematics Attitudes	3.63 (1.74)	3.67 (1.84)	3.65 (1.75)	5.36 (1.21)	5.25 (1.15)	5.34 (1.18)	5.10 (1.63)	4.43 (1.51)	4.69 (1.56)	4.95 (1.56)	4.24 (1.64)	4.62 (1.63)

Table 1. Mean (and standard deviations in parentheses) of participant background characteristics by major and gender.

Testing Instruments and Psychometric Properties

A number of tests were used to illustrate participants' background, experience, and abilities. Efforts were made to assess the psychometric properties of the instruments used, thus as the very least, Cronbach's alpha is provided for each applicable test.

Introduction to Stata

Stata is a commercially available statistical package designed for research professionals such as economists, political scientists, epidemiologists, and other health scientists. Stata was chosen as a typical end-user programming environment because of its traditional textual command line interface and limited environmental support for programming and debugging. A typical user view of Stata can be seen in Figure 1. Stata for Windows was used in the study.
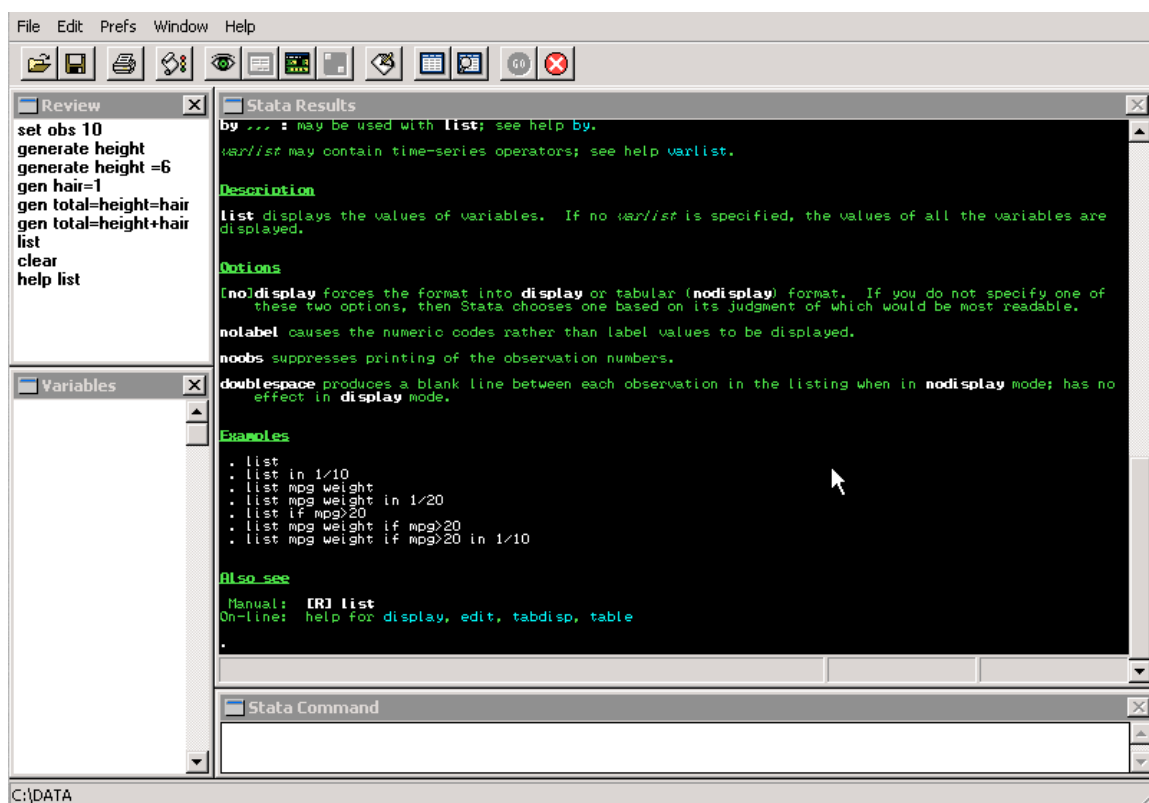
Figure 1. A typical user view of Stata.

In terms of graphical user interfaces, Stata provides a main window which contains a *variables* window, which shows the variables in the current data set being used, a *review* window that shows a history of user-generated and computer-generated commands, an *output* window, that shows a history of textual output generated by Stata, and a command window in which users enter textual commands. When graphs are generated, Stata displays the graph in a simple resizable, non-interactive window. Stata also provides a spreadsheet-like data editor with limited functionality (see Figure 2).
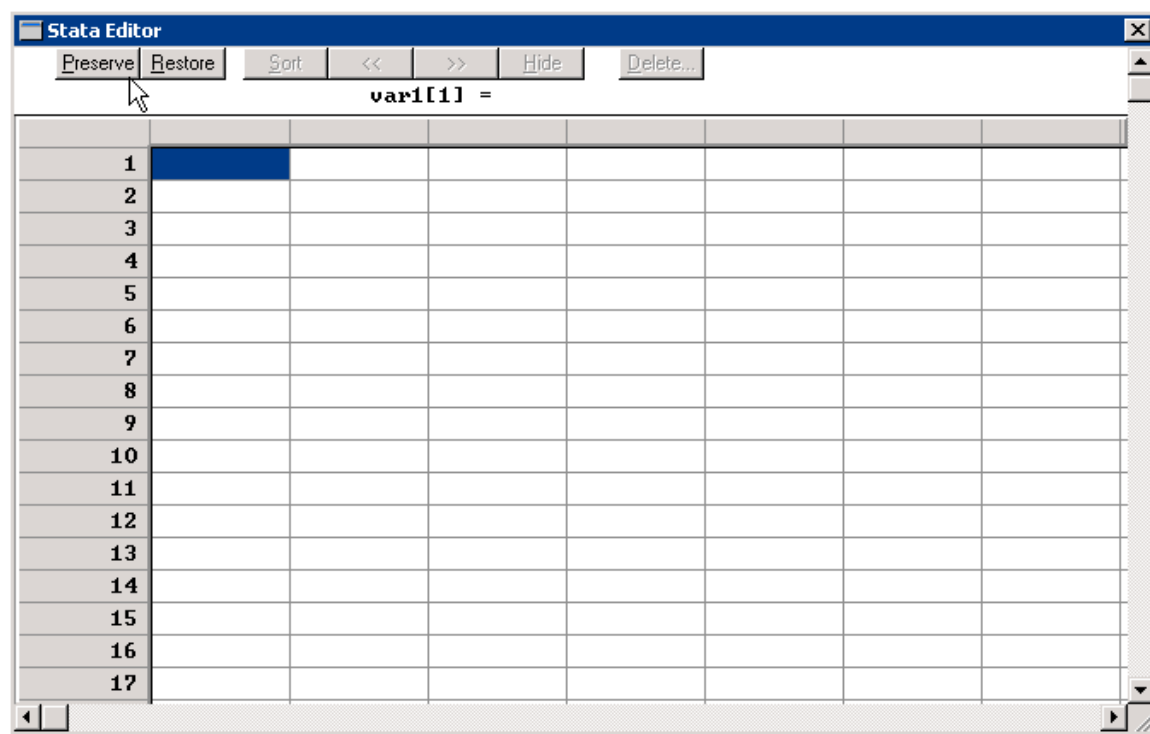
Figure 2. The Stata editor window, providing a spreadsheet-like
   interface for inserting and deleting data and variables.

Creating a data set in Stata consists of setting a system variable that specifies the

number of objects in a data set to the desired size; such a command would look like "set

obs 1000," for a data set consisting of 1000 objects. To create variables within this data

set, the *generate* command is used; for example, the command "generate height = 5"

would create a variable named *height*, which has the value 5 for all 1000 objects. Stata

provides a number of functions that can be used in conjunction with the generate

command; for example, the command "generate height = uniform()" creates a variable

named height in which each observation is a value between 0 and 1 selected from a

uniform distribution of random numbers.

Stata provides an online help system with descriptions and limited examples of

command use. The online help can be reached through the commands "help" or

"search"—in which case help is displayed in the *output* window or through the help menu

in the main Stata window, in which case the help text is displayed in a graphical,

resizable window. See Figure 3 and Figure 4 for examples of each interface respectively.

Regardless of the interface used, a request for help for the word p-value, for example,

would return an error message "help for p-value not found. Try help contents or search

p-value." A search on text will return a list of matches, akin to a web search (see Figure

4), while a failed search will provide no feedback unless the search in the menu is used.

See www.Stata.com for more information about Stata.

Figure 3. Textual help in the output window for the t-test command.
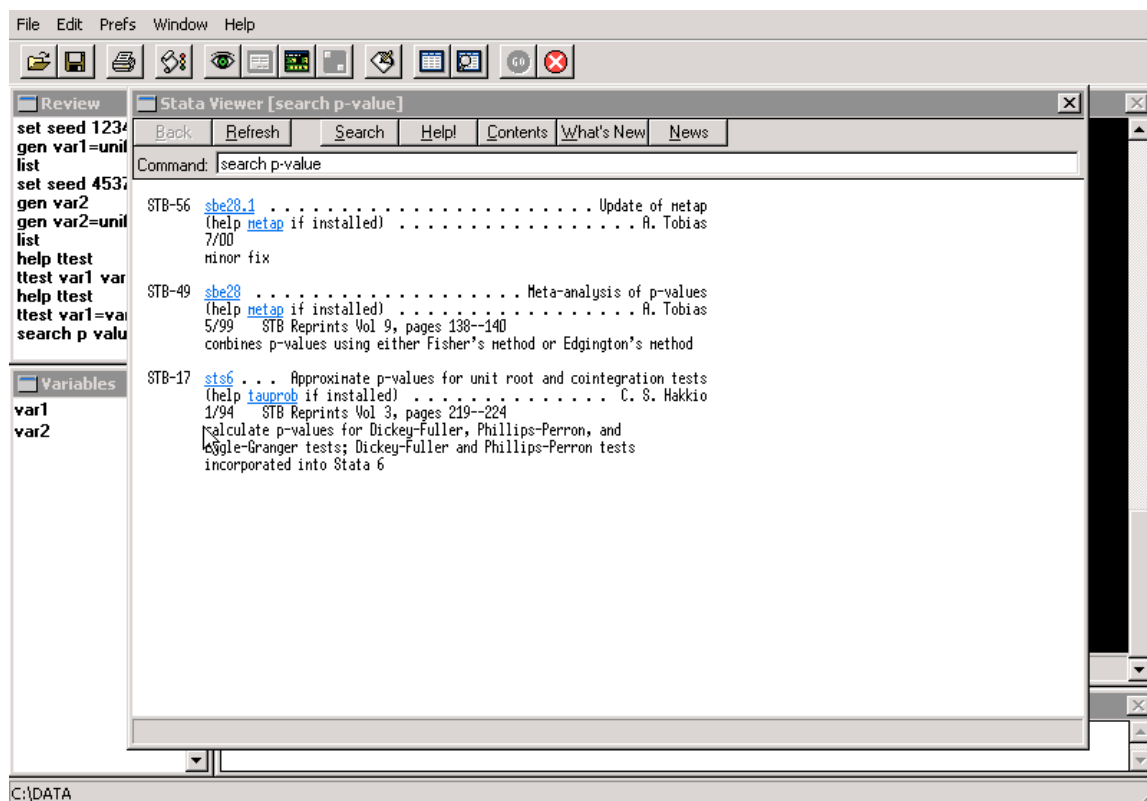
Figure 4. A view of the graphical help window, showing search results on the keyword 'p-value'

Vocab27

The Vocab27 test consisted of 27 multiple-choice questions that present a word and asked the participant to choose the word from the list of choices that has the most similar definition. The test was intended to be a coarse measure of IQ. No time limit was placed on the test, but if participants took more than 20 minutes, the experimenter asked them to finish. This only occurred twice in the sample. Cronbach's alpha for this sample was .75, which suggests fairly good reliability.

VCog I

Vcog I consisted of 50 items, equally divided into five sections of 10 items each. In the first section, the participant was asked to determine the next number in a series of numbers. For example, the sequence "1, 2, 4, 8, __" would be provided and the participant would fill in the next number in the series. In the second section, participants were given a word pair that has an unidentified relationship and are asked to choose two words from a list that have the same relationship. For example, the word pair "house, window" would be presented, and the participant would circle two words from the list "body, x-ray, engine, screw, knife" that have the same relationship; the correct answer in this case would be body and x-ray, since an x-ray lets one see into a body just like a window lets one see into a house. In the third section, participants were asked to choose two words from a list that have the most similar meaning; for example, participants would choose two words from the list "home, hole, house, car, cave," the right answer in this example being house and home. In the fourth section, participants choose two words form a list that have the most opposite meanings; for example, from the list "berate, argue, rebuke, yell, stomp" the correct answer would be berate and rebuke. In the final section, participants were given a sequence of pictograms and asked to choose the next pictogram in the series from a list. Patterns in this final section were based on the rotations and reflections of pieces of the pictograms.

The test was intended to be a measure of verbal, spatial, and mathematical ability, and was used in this study to measure the general intelligence of the participants. Participants were given 20 minutes to complete the test and were told when they had 10 minutes

remaining. Cronbach's alpha for this sample was .78, suggesting a solid reliability for the test.

Statistics Test

The statistics test, created specifically for this study, consists of 10 multiple-choice questions (varying from three to five choices) that covered material from typically offered in introductory statistics and hypothesis testing courses. An initial item pool of 20 items was formed from publicly available statistics tests administered by instructors in the OSU Department of Statistics. One example item from the test is given in Figure 5. The original set of items was administered to three volunteers who were asked to identify confusing and difficult items. The final 10 items were chosen and revised according to these early administrations. Participants were given 10 minutes to complete the test and were told when they had five minutes remaining.

Ignoring twins and other multiple births, assume babies born at a hospital are independent events with the probability that a baby is a boy and the probability that a baby is a girl both equal to 0.5. The probability that the next five babies are girls is

a.   0.00098
b.   0.03125
c.   0.25
d.   0.5
e.   1.0

Figure 5. A question from the statistics test given to participants to
   measure knowledge of statistics.

Cronbach's alpha for this test was .21, which by all standards condemns the usefulness of the measure. As can be seen in Table 1, there was a very low ceiling placed on the test, resulting in very poor performances. Despite the poor psychometric properties of the test and since the questions for this were taken from introductory statistics courses, we can

conclude with some confidence the sample as a whole has a poor understanding of statistical concepts.

Background Questionnaire

The background questionnaire was administered to gather basic information about participants' age, major, native language, as well as participant-reported experience and ability with statistics, mathematics, computer software, and programming. The questionnaire also measured participants' attitudes towards these areas of knowledge and the experiment itself. Each of the experience and attitude questions followed the typical Likert format and can be seen in Figure 6, and Figure 7, respectively. Participants were not given a time limit for this questionnaire.

I have a lot of difficulty doing algebra	1	2	3	4	5	6	7
I have a lot of difficulty doing trigonometry	1	2	3	4	5	6	7
I have a lot of difficulty doing geometry	1	2	3	4	5	6	7
I have a lot of difficulty doing calculus	1	2	3	4	5	6	7
I am very effective in using spreadsheet software	1	2	3	4	5	6	7
I am very effective in using statistical software	1	2	3	4	5	6	7
I am very effective in using word processors	1	2	3	4	5	6	7
I am very effective in using database software	1	2	3	4	5	6	7
I am very effective in using internet browsers	1	2	3	4	5	6	7
I am very effective in using StatGraphics	1	2	3	4	5	6	7
I am very effective in using Stata	1	2	3	4	5	6	7
I am very effective in using StatView	1	2	3	4	5	6	7
I am very effective in using SPSS	1	2	3	4	5	6	7
I have a lot of experience programming in C	1	2	3	4	5	6	7
I have a lot of experience programming in C++	1	2	3	4	5	6	7
I have a lot of experience programming in Java	1	2	3	4	5	6	7
I have a lot of experience programming in JavaScript	1	2	3	4	5	6	7
I have a lot of experience programming in BASIC	1	2	3	4	5	6	7
I have a lot of experience programming in HTML	1	2	3	4	5	6	7
I have a lot of experience programming in Lisp	1	2	3	4	5	6	7
I have a lot of experience programming in Visual Basic	1	2	3	4	5	6	7

Figure 6. The experience questions asked on the background questionnaire. The scale included "I've Never Used It" on the far left, and from "Strongly Disagree" to "Strongly Agree" from left to right.

I am excited about doing this experiment	1	2	3	4	5	6	7
I enjoy using computers	1	2	3	4	5	6	7
I enjoy doing statistics	1	2	3	4	5	6	7
I struggle with mathematics	1	2	3	4	5	6	7
I enjoy or think I would enjoy computer programming	1	2	3	4	5	6	7
I am a good mathematics problem solver	1	2	3	4	5	6	7
I am a good computer problem solver	1	2	3	4	5	6	7
I enjoy using statistics to test hypotheses	1	2	3	4	5	6	7
I am nervous about this experiment	1	2	3	4	5	6	7
I think this experiment will be boring	1	2	3	4	5	6	7
I consider computers a useful tool	1	2	3	4	5	6	7
I am a good statistics problem solver	1	2	3	4	5	6	7
I am comfortable around this experimenter	1	2	3	4	5	6	7
I enjoy doing mathematics	1	2	3	4	5	6	7
I will not use statistics in my everyday life	1	2	3	4	5	6	7
I have confidence in my math abilities	1	2	3	4	5	6	7

Figure 7. The attitude questions asked on the background questionnaire, ranging from "Strongly Disagree" to "Strongly Agree", left to right.

Experience and ability self-reports were divided into four categories: software, mathematics, statistics, and computer programming. Cronbach's alpha for each of these scales respectively was .67, .97, .60, and .80, suggesting that each of these groups of questions form relatively strong measures of experience. Attitude questions were divided into four categories: computers, statistics, mathematics, and the experiment. Cronbach's alpha for each of these scales respectively was .80, .76, .93, and .60.

Post-Session Questionnaire

The post-session questionnaire was administered immediately following the completion of the second problem and was intended to gather information about the participants experience with the software (see Figure 8) and their perception of the strategies they used to solve the problems and learn to use Stata (see Figure 9). Six free response questions were also included in order to gather anecdotal data on their experiences. The data from the post-session questionnaire was mainly used to confirm the validity of the data extracted from user interactions.

I enjoyed using this software	1	2	3	4	5	6	7
I am confident that I could learn to use this software	1	2	3	4	5	6	7
I was comfortable using the software	1	2	3	4	5	6	7
I was confused about how to use the software	1	2	3	4	5	6	7
I was curious about how to use the software	1	2	3	4	5	6	7
I was frustrated with the software	1	2	3	4	5	6	7
I didn't know how to learn about the software	1	2	3	4	5	6	7
The software is intimidating	1	2	3	4	5	6	7
The software was difficult to use	1	2	3	4	5	6	7
Using the software was intuitive	1	2	3	4	5	6	7

Figure 8. Questions about participants' experience with Stata following the problem solving sessions. The scale ranged from "Strongly Disagree" to "Strongly Agree", left to right.

Used resources within Stata, such as help and search	1	2	3	4	5	6	7
Wrote down thoughts and ideas on scratch paper	1	2	3	4	5	6	7
Drew pictures on scratch paper to help me think	1	2	3	4	5	6	7
Thought to myself	1	2	3	4	5	6	7
Began to learn about one aspect of Stata, then broke it down into smaller pieces to learn individually	1	2	3	4	5	6	7
Learned about smaller aspects of Stata, then combined what I learned to understand larger aspects	1	2	3	4	5	6	7
Guessed about how to do something, tried it, failed, and repeated until I was successful	1	2	3	4	5	6	7
Explored the Stata program without any strategy in mind	1	2	3	4	5	6	7
Took a specific case in Stata and generalized about how something worked	1	2	3	4	5	6	7
Used similar examples to generalize to the task I needed to perform	1	2	3	4	5	6	7
Clarified or rephrased information that I found about Stata	1	2	3	4	5	6	7
Used simpler cases I found in Stata to understand more complicated cases	1	2	3	4	5	6	7

Figure 9. Questions asked following the problem solving sessions about the strategies participants used to solve the problems. The scale ranged from "Strongly Disagree" to "Strongly Agree", left to right.

Tutorial

Participants were given a 10-minute tutorial on how to write Stata commands, how to create a data set of a certain size, how to create a variable, how to list the data in a data set, and how to get help from the Stata command line. Rather than provide the knowledge necessary to succeed at the two problems, the intention of the tutorial was to allow participants to become accustomed to the environment and the basic features the environment provided. The philosophy was that the participants should use their own strategies to learn about the environment rather than strategies the experimenter could provide, in order to facilitate the observations of the strategies participants would use in a natural setting.

The experimenter followed a tutorial script, provided in Figure 10, to ensure that each participant received the same instructions. Participants were told they may ask questions during the tutorial, but only regarding the topics covered in the tutorial itself. If participants asked questions about material not covered in the tutorial, they were told to investigate their question during the problem solving sessions. The tutorial typically lasted about 10 minutes.

- As we go through this tutorial, if you don't understand something, of if you need me to slow down, please interrupt me.
- Before we use the computer, let's first do a brief review about statistics by starting with a scenario.
- Suppose you're interested in the differences in the height and hair length of men and women.
- We can use statistics to tell us whether or not there are differences in the average height and average hair length between men and women. These days, we can use statistics software like Stata to help us do this.
- The basic idea behind any statistics software like Stata centers on three concepts, which we can see in our scenario.
- The first is that of the objects that we observe and make measurements on. In our scenario, these objects are people. Assume that we've recruited 10 people and measured their height and hair length. How would we begin to analyze the data with Stata?
- Go ahead and wake up the computer, and open up Stata. Find the Stata command window at the bottom of the screen. *[wait for participant to complete task]*
- Stata begins with 0 objects, so to tell Stata that we have 10 objects, or people, we type "set obs 10" and press enter.
- Notice that some text appeared in the window in response to your command, telling you that the number objects was 0 and has been set to 10. Do you see the green text in the black area? *[wait for participant response]*
- Also notice that the command you typed appeared in the upper left window entitled "review." You can use this window to review the commands that you've entered and remind yourself what to type later.
- The second concept is that of a *variable*. In our scenario, we have two variables: height and hair length. Let's say that the ten people we made measurements on were very similar, and all were 6 feet tall. To tell Stata this information, we need to type a command that creates data for each object. So let's type "generate height = 6" and press enter. *[wait for participant to enter command]* Notice this makes a variable called "height" and it lists this below the review window in the window called variables. Now, Stata knows that for all ten people, the variable "height" is equal to 6.
- What about hair length? Let's say that since they were all very similar, that all had a hair length of 1 foot. This time, instead of typing the whole word, we can type "gen hair = 1" and press enter. Now, a new variable named "hair" has appeared in the "variables" window, along with "height", and now, Stata knows that for all ten people, the variable "hair" is equal to 1.
- Let's say we were also interested in their total height-the height of their body plus the length of their hair. To create a variable that stored these numbers, type "generate total = height + hair" and press enter. *[wait for participant to enter command]* Now, there is a variable called "total" that equals the height plus the hair length of each person we made measurements on.
- To see all of this information, we can type the command "list" and press enter and Stata will show a table of all three variables and all ten people. *[wait for participant to enter command]*
- By looking at this table, we can see the third concept, which is that of an *observation*. Notice that for person 1, we have three *observations*: person 1's height, person 1's hair length, and person 1's total height including hair length.
- Also notice in this table, that for the variable hair, we have ten *observations*: one hair length measurement for each person. So the concept of an *observation* is really just that of a single measurement, such as the hair length of person 6, or the total height of person 2.
- Although Stata has many more commands that allow you to do many things, such as calculate the mean hair length or graph the hair length versus the height, our data isn't very interesting. Let's clear the system of all of our observations, variables, and people by typing "clear" and pressing enter. *[wait for participant to enter command]*
- This sets the number of objects to 0 and removes all of the variables.
- If you ever need help with something, you can use the commands "help" and "search" in the Stata command window, followed by a word, and Stata will give you help on that word. For example, let's see the help on the command "list" by typing "help list."
- To continue scrolling through the help, simply press the space bar. Go ahead and go all the way down to the bottom.
- Let's go back into the help for the list command and practice quitting. *[if they need help, remind them to type "help list"]* So to quit the help, press 'q'. The "break" in the output window means that you have quit the help.
- Lastly, also notice there is also a help menu at the top of the Stata window in the menu bar.

Figure 10. The tutorial script followed by the experimenter, introducing the participants to basic features of Stata's environment. Italicized statements surrounded by square brackets were instructions for the experimenter and were not read aloud.

Problems

Participants were given two problems in the experiment, intended to evaluate their ability to construct commands and to debug commands written by someone else.

Problem 1. The first asked participants to created a data set of 1000 objects, create two variables with uniformly distributed random numbers between zero and one, to perform a two-sided unpaired t-test to compare the means of the two lists of data, and to report the p-value returned by the t-test to the experimenter (see Figure 11 for the description participants received). The solution consisted of a command to set the number of objects to 1000 ("set obs 1000"), two commands that used the *uniform* function in conjunction with the generate command ("generate var1 = uniform()"), and the use of the t-test command to compare the two lists of data ("ttest var1 = var2, unpaired"). Participants were given 30 minutes to complete the problem and were told that they would not receive their extra credit if they gave up with time remaining. There was no consequence for running out of time.

Problem 1

Using STATA, perform the following tasks:
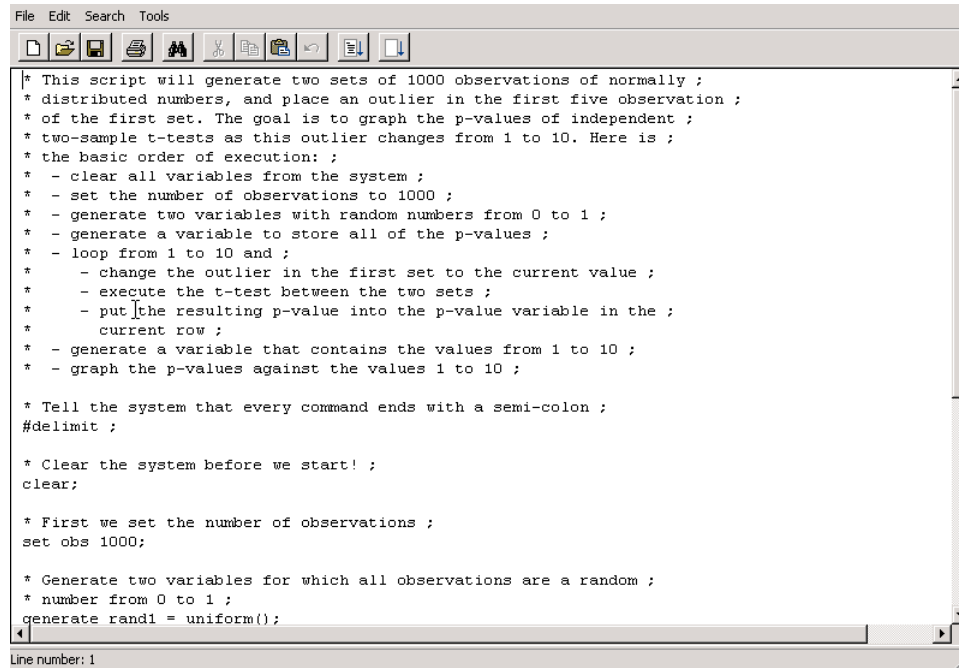
- Create a data set with 1000 objects.
- For this data set, create two variables that each has a uniformly distributed set (and specifically uniform, not normal or any other type of distribution) *[ask participant "do you know what I mean by uniform?" and define it if not using the statement "you probably know what a bell curve is; there are many data points at the mean and not many at the edges; a uniform distribution is one where there are just as many of one number as there are another, just like a flat histogram."]* of random numbers between 0 and 1. *[Ask participant "do you know what I mean by random?" and define it if not using the statement "random means that every number is different and there are no patterns"]* In other words, you will have two sets of 1000 numbers that all have different numbers such as 0, 0.235 or 0.834. *[Explain to the participant that this means they will have 2000 numbers in all, across to variables]*
- With this data, perform a two-sided, unpaired t-test on the two variables to help determine if their means differ. *[Ask participant "Do you know what a t-test is?"]* A t-test is a statistical equation that will suggest whether or not the means of two sets of numbers are significantly different. The t-test will give a number called a p-value, which is between 0 and 1. *[Remind participant of the meaning of these numbers: "if the two sets are really different, the p-value is very close to zero; if they are very similar, the p-value is closer to one"]*
- Report the p-value to Andy.

YOU HAVE 30 MINUTES. REMEMBER TO think aloud!

Figure 11. The description of problem 1 that participants received. Italicized statements surrounded by square brackets were read aloud by the experimenter by were not included on participant copies.

Problem 2. In the second problem, participants were given a short sequence of Stata commands, called a *do-file* in Stata (see Figure 12 for a view from Stata, and see Figure 13 for a listing of the do-file text). The purpose of the do-file was to create a graph that visualized the influence of increasingly large outliers on the p-value from a t-test. The *do-file* created a data set of 1000 objects with two variables with random values between zero and one, and looped ten times, each time changing five of the values in the first variable to increasingly large outliers from 1 to 10. As these outliers increase from 1 to 10, the means of the two lists of data should become increasingly different, and thus the p-value from the t-test between these two lists should become smaller. The graph listed the outlier values from one to ten on the x-axis, and the p-value generated by the t-test for each value of the outliers on the y-axis. See Figure 14 for a listing of the problem description that participants received.

```
File   Edit   Search   Tools

* This script will generate two sets of 1000 observations of normally ;
* distributed numbers, and place an outlier in the first five observation ;
* of the first set. The goal is to graph the p-values of independent ;
* two-sample t-tests as this outlier changes from 1 to 10. Here is ;
* the basic order of execution: ;
*  - clear all variables from the system ;
*  - set the number of observations to 1000 ;
*  - generate two variables with random numbers from 0 to 1 ;
*  - generate a variable to store all of the p-values ;
*  - loop from 1 to 10 and ;
*     - change the outlier in the first set to the current value ;
*     - execute the t-test between the two sets ;
*     - put the resulting p-value into the p-value variable in the ;
*        current row ;
*  - generate a variable that contains the values from 1 to 10 ;
*  - graph the p-values against the values 1 to 10 ;

* Tell the system that every command ends with a semi-colon ;
#delimit ;

* Clear the system before we start! ;
clear;

* First we set the number of observations ;
set obs 1000;

* Generate two variables for which all observations are a random ;
* number from 0 to 1 ;
generate rand1 = uniform();
```

Line number: 1

Figure 12. The participants' initial view of the do-file used for problem 2.

```
* This script will generate two sets of 1000 observations of uniformly ;
* distributed numbers, and place an outlier in the first five observations ;
* of the first set. The goal is to graph the p-values of unpaired ;
* two-sample t-tests as this outlier changes from 1 to 10. Here is ;
* the basic order of execution: ;
*  - clear all variables from the system ;
*  - set the number of observations to 1000 ;
*  - generate two variables with random numbers from 0 to 1 ;
*  - generate a variable to store all of the p-values ;
*  - loop from 1 to 10 and ;
*     - change the outliers in the first set to the current value ;
*     - execute the t-test between the two sets ;
*     - put the resulting p-value into the p-value variable in the ;
*        current row ;
*  - generate a variable that contains the values from 1 to 10 ;
*  - graph the p-values against the values 1 to 10 ;

* Tell the system that every command ends with a semi-colon ;
#delimit ;

* Clear the system before we start! ;
clear;

* First we set the number of observations ;
set obs 1000;

* Generate two variables for which all observations are a random ;
* number from 0 to 1 ;
generate rand1 = uniform();
generate rand2 = uniform();

* Next, we need to generate a variable to store the p-values in. ;
* We put 0's in every observation for now. ;
generate pvalues = 0 in 1/100;

* Then we need to loop through outlier values from 1 to 10,;
* at intervals of 1. First, we update the first five observations ;
* of variable rand1. Then, we run an unpaired t-test between the ;
* two variables. Finally, we place the p-value in the pvalues ;
* variable, in the current row. ;
for num 10/1: replace rand1 = X in 1/5 \ ttest rand1 = rand2, unpaired \ replace pvalues = r(p) in X;

* Generate a variable that contains 1 through 10 ;
generate valueofoutlier = _n in 1/10;

* Graph the results, pvalues versus valueofoutlier ;
graph valueofoutlier pvalues, title(p-values of a t-test as the magnitude of outliers increase) connect(l);
```

Figure 13. The source code provided to participants for problem 2. The four bugs are highlighted in grey.

Problem 2

This do-file graphs the influence of having several increasingly large outliers in your data set; *[ask participant if they know what an outlier is; if not, use this statement to describe it: "an outlier is like a ten foot man in a group of six foot mean; his height doesn't fit with the rest of the data; it stands out and influences the mean by making it much larger than it should be"]* it plots the p-value calculated by the t-test of two variables as you change five observations in one of the variables into outliers (in real-life data sets, such outliers are often caused by data entry errors). The example highlights undue influence on your conclusions due to a few outliers. The do-file does this demonstration in the following way:

- It makes two variables with 1000 observations of random numbers between 0 and 1, just like in the previous problem.
- As you recall from the previous problem, the means of the two variables should be fairly equal when there are no outliers, and so the p-value of a t-test should be closer to 1 than to 0. *[Instruct the participant to look at the table in the output window in Stata from the previous problem, and observe that the t test on the two data sets, which had very similar means, is closer to zero]*
- The do-file is supposed to change the first five observations of the first variable into increasingly large outliers. The change is done in a series of 10 steps: for the first step, the outliers are set to 1, for the second step the outliers are set to 2, and so on until the last step when the outliers are set to 10.
- At each step, after the observations are changed, a t-test is performed and the p-value is stored.
- When the value of the outlier put into the first variable is 1, the means of the two variables should be pretty close. But when the value of the outlier put into the first variable is really high, like 10, the means should be pretty different, because the outlier makes the mean bigger.
- As the outlier gets bigger, the means of the two variables get more different, and so the p-value from a t-test should become smaller, indicating that the means differ more and more.
- The resulting graph should look something like this: *["like something on your paper"]*
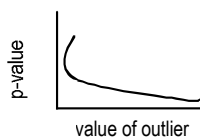
- But, the do-file has many problems! There are many things it's supposed to do, as described above, that it doesn't do correctly!
- Analyze the do-file, find as many of the problems as you can, and fix them. None of the problems are major ones, and only require small changes. When you think you've fixed all of the problems and you think you see the correct results, let Andy know. *[Explain to the participant: "So basically, this file does everything on this description; there is no need to create the commands to perform this operations]*

YOU HAVE 20 MINUTES. REMEMBER TO think aloud!

Figure 14. The description of problem 2 that participants received. Italicized statements surrounded by square brackets were read aloud by the experimenter by were not included on participant copies.

The resulting graph was supposed to be a smooth curve with ten data points as shown in Figure 15. However, four bugs were inserted into the *do-file* that the participants received (highlighted in Figure 13) which changed the graph produced. The effect of these four bugs was a graph with inverted axes and 90 extra p-values with values of zero (see Figure 16). Participants were told to find out what was wrong with the *do-file* and

change the commands in it so that the correct graph was produced; once they thought they had the correct graph and had found all of the bugs they were to notify the experimenter. Participants were given 20 minutes to complete the problem and were told that they would not receive their extra credit if they gave up with time remaining. There was no consequence for running out of time.
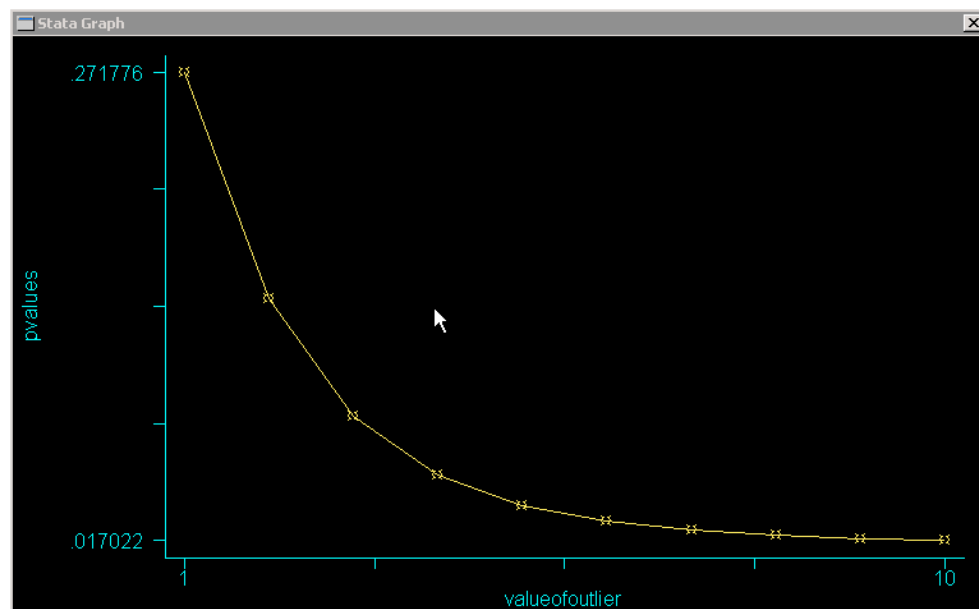
Figure 15. The graph that the do-file was supposed to create.

Each of the bugs had a specific rationale:

- The bug starting on the line "for num 10/1: replace rand…" simply changed the order of execution of the iteration, which did not affect the data or the graph. This bug was inserted to see if participants would exert effort to change a bug that did not corrupt the output of the do-file.

- The bug in the line "generate pvalues = 0 in 1/100;" only required the "1/100" to be changed to "1/10," because only ten p-values were being calculated. The extra p-values cause the vertical line in Figure 16 to appear. This bug required an understanding of the syntax in "1/100." This bug was contrasted with the

bug in the line "generate valuesofoutlier = 0 in 1/100;" in which the only

difference is that the latter command is accompanied by a comment above it

that specifically says "generate a variable that contains 1 through 10." These

bugs were inserted to observe the magnitude to which the comment would aid

participants in identifying the bug.

- The final bug was in the last line of the file, which graphs the data. The two

    variables listed in the command are reversed: the pvalues variable should come

    before the valueofoutlier variable, as stated in the comment above the

    command. As the only way to detect this bug was to inspect the output of the

    program, it was inserted to see the degree to which participants would analyze

    the graph the do-file produced.
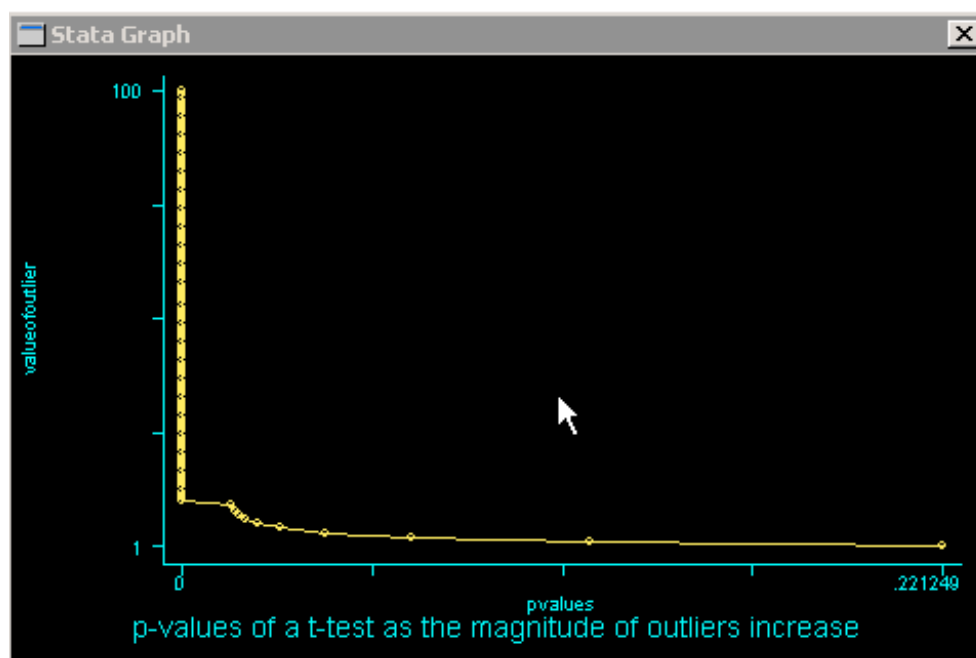
Figure 16. The graph produced by the do-file in problem 2 without any
modifications. The straight vertical line comes from the zeros in the
pvalues variable.

Experimental Setting

Participants were tested in one of three environments, two of which were vacant faculty offices and one of which was a small computer lab. All three environments housed multiple personal computers, windows, and chairs. Throughout each session, only the experimenter and the participant were present in the room. Experiment sessions occurred through the summer and fall months of 2002.

Procedures

An experiment protocol was written for the experimenter to follow so that each participant received the same information and in the same order; though it is not listed here, it is summarized in detail. Participants were tested individually and arrived at the beginning of a scheduled 2-hour block. Participants were asked to sit down in the chair in front of their personal computer and the experimenter briefly described the purpose of the experiment. Following any questions the participant had about the experiment, the experimenter administered the Vocab27 test notifying the participant of the 10-minute time limit. After this test, the experimenter administered the VCogI test, notifying the participant of the 20-minute time limit; the experimenter cued the participant after 10 minutes had elapsed. Finally, the experimenter administered the Statistics test and notified the participant of the 10-minute time limit; the experimenter cued the participant after 5 minutes had elapsed.

Following the battery of tests, participants were given the option of a 5-minute bathroom break. Following the optional break, participants were given the background

questionnaire and told there was no time limit. Once they completed the questionnaire, the experimenter began the Stata tutorial, which covered the creation of a data set, of variables, and how to find help within Stata. Participants were given the opportunity to answer questions about anything covered in the tutorial, but told to avoid other questions.

Next, the experimenter reminded the participant that they would be working on two problems within Stata and that there would be three rules regarding the problem solving sessions: (1) they were not allowed to ask the experimenter questions, (2) they were not allowed to use the Internet to solve their problems, and (3) they were to work until they solved the problem or time expired. Participants were told that if they gave up, they it "would be considered not completing the problem and [they] would suffer whatever consequences there were." Once the participants expressed their understanding, the experimenter gave the participant a copy of problem one's description and read it aloud. Participants were allowed to ask for clarification while the description was read, and the experimenter also offered clarification if he sensed it was necessary. Once the problem was understood, participants began problem one once the experimenter started the recording devices.

During the problem solving session, if questions were asked the experimenter would answer "I'm sorry but I can't answer your questions" except in three circumstances: the participant asked how much time remained, asked the experimenter to clarify some text in the problem description, or asked the experimenter to clarify one of the three rules regarding the problem solving sessions.

Following problem one, participants were given the solution (seen in Figure 17) if the participant had not completed the problem in its entirety. If they completed part of the

problem, the experimenter started from that point in the solution. The effect of providing

the solution to those participants who failed was hopefully to bring the level of

understanding about the environment to that of those participants who succeeded.

Next, problem two's description was given to the participant, and it too was read

allowed, during which clarification was offered. Once the problem was understood,

participants began problem two once the experimenter started the recording devices.

- Set the number of objects to 1000
- Search for help on generating random numbers
- Go the help for generate, which is the third search result
- Read the description of the generate command
- Follow the link to the help for functions
- Read the description of the uniform function
- Read the introduction in the functions help on how to use functions
- Generate two variables following the syntax examples
- Search for help on t-tests
- Follow the third search result for help on the ttest command
- Follow the example in the help for the ttest command
- Choose the appropriate p-value

Figure 17. The most efficient solution to problem 1, given to
participants following the first problem solving session. After each
instruction was given, participants were given help when necessary
and questions about the solution were answered.

Once the participant was done with problem two or time expired, the solution was

offered if the participant desired it and the post-session questionnaire was administered

with no time-limit. Regardless of whether the participant had given up on either problem,

their name was entered in the raffle on the condition that details about the experiment

would not be shared with classmates.

Data Acquisition

Participants used a desktop PC with Windows 2000, a 17" monitor, and a basic two-

button mouse and keyboard. The screen resolution was set to 800 by 600 pixels and color

depth to 256 colors. During the extent of the problem solving sessions, the only window

visible to participants was the main Stata window.

During each of the problems, screen capturing software was used to capture

participants interaction with Stata at 12 frames per second at the same resolution that

participants viewed the screen in (800 by 600 pixels); the small screen configuration was

chosen so that the screen capture software would not interfere with participants'

interaction with Stata. In order to capture audio and interactions with the external

environment, a Sony Digital8 camcorder was used to videotape participants over the

shoulder. A microphone was attached to the monitor in order to get a clear recording of

what participants said during the problem solving sessions. The videotape was digitized

and, along with the screen capture videos, burned onto compact disc.

The non-quantitative videotapes and screen captures went through two phases of

quantification that are described in detail below.

Coding Procedures

In order to convert the videotape and screen captures into a form that could be

quantified, a coding scheme was developed to capture and describe the fundamental

aspects of interaction with Stata and used to create transcripts for each participant's

interaction. The coding scheme is listed in Table 2. Each row represents an action that the

participant could perform in external environment or within Stata, while the italicized

words following the action represent the information that was coded in participant

transcripts; the *source* field typically includes the window the action was performed in,

the *info* field typically includes information that was provided by the user, and the *context*

field typically includes contextual information about the action. A four-digit string that

represented the number of minutes and seconds into the interaction also preceded each

action. For example, if a participant moved the help window off screen five minutes and

three seconds into the problem, the action "0503 WIN H MV <off the screen to the

right>" would be recorded. An excerpt from a transcript is provided in Figure 18.

Action	Source	Info	Context	Description of the Action
		Actions Extracted from Videotape (External Environment)		
U		*words*		Participant speaks *words*
ME		*words*		Experimenter speaks *words*
WR		*words*		Writes *words* on paper
RE				Reads the problem description
DR		*description*		Draws *description* on paper
EX	*window*			Examines *window*
		Actions Extracted from Screen Capture		
WIN	*window*	*window operation*	*<context>*	Performs *window operation* on *window*
BAR	*window*	*direction*	*<context>*	Scrolls in window in *direction* near *<context>*
MEN	*window*	*menu name*	*menu item*	Clicks on *menu item* in *menu name* menu in *window*
BUT	*window*	*button name*		Clicks button *name* in *window*
CUR	*window*	*text*	*<context>*	Points to *text* with cursor in *window* near *<context>*
LINK	*window*	*link*	*<context>*	Clicks *link* in *window* near *<context>*
INS	*window*	*text*	*<context>*	Inserts *text* from *window* near *<context>*
DEL	*window*	*text*	*<context>*	Deletes *text* from *window* near *<context>*
SEL	*window*	*text*	*<context>*	Selects *text* from *window* near *<context>*
COP	*window*	*text*	*<context>*	Copies *text* from *window* near *<context>*
PAS	*window*	*text*	*<context>*	Pastes *text* to *window* near *<context>*
CUT	*window*	*text*	*<context>*	Cuts *text* from *window* near *<context>*
H	*window*	*keyword*		Gets help on *keyword* from *window*
S	*window*	*keyword*		Searches on *keyword* from *window*
SP			*<context>*	Presses the space or enter key to scroll output to *<context>*
Q			*<context>*	Quits the help in the output window near <context>
RUN				Runs a do-file by clicking on the run button
C		*keystrokes*		Types *keystrokes* in command window
COM		*command*		Enters *command* in command window
ERR		*error*	*<context>*	Error message *error* becomes visible, described by *<context>*
		Miscellaneous Actions		
ST		*comment*		A comment on the participant's state by the coder
START		*problem*		Starts problem number *problem*
STOP		*problem*		Stops problem number *problem*
		Legend		
window		One of the following windows in Stata: REV (review), OUT (output), VAR (variables), DO (do-file), COM (command), GRA (graph), H (help), STA (main Stata), DIA (dialog box), ED (editor)		
window operation		C (close), MIN (minimize), MAX (maximize), RE (resize), MV (move), FR (bring to front)		
direction		U (up), D (down), L (left), R (right)		
keystrokes		Keystrokes recorded in the "C" action included the '~' character, which represents a backspace, and the '@' character which represents a keystroke of the enter key, which executes the command.		

Table 2. A list of the actions that were coded for the videotape and
  screen captures for each participant.

```
0401 WIN DO FR
0403 BAR DO L <back to left of the file>
0407 BAR DO R <to see the right of the screen>
0412 BAR DO L <back to the left of the file>
0423 CUR DO <the gen valueofoutlier command and comments>
0431 CUR DO replace rand1 = X in 1/5 <in for loop>
0432 BAR DO R <to see rest of for loop>
0434 BAR DO L <back to left of file>
0440 DEL DO 0 <to make gen pvalues 1/10 instead of 1/100>
0444 RUN
0445 WIN GRA FR
0445 ST The graph looks good except for axes, and one p value dips down a bit.
0450 CUR GRA <traces the curve>
```

Figure 18. An excerpt from a participant's transcript, portraying the information that was coded from screen captures and videotape.

The process of coding was performed by two individuals trained to comply with the coding scheme defined in Table 2. After the videotape was digitized, the two coders followed this procedure to create the transcripts:

- Code the external environment actions from the digitized videotape. The majority of these actions were users own words but care was taken to observe when participants' attention moved from the computer monitor to paper materials in the environment.

- Code the Stata environment actions from the screen captures. In order to ensure the accuracy of the timestamps, the screen capture videos were advanced manually and stopped each time the participant performed an action.

If at any time a coder perceived that the participant misunderstood the goal of the problem or was severely inhibited by a lack of English comprehension, the coding was discontinued and the two coders came to consensus on whether or not the participant was confused. If the two coders agreed on confusion, the participant's data was dropped from the study; in no cases did the coders disagree. As mentioned earlier, 11 participants were dropped from the study for this reason. The coders each reported an accuracy of time

stamps of within 2 seconds. One coder coded 16 of the 75 participants' interactions and the other coded 61.

After transcripts were created for each participant, two separate transcripts existed for each participant: one containing videotape actions and one containing screen capture actions. In order to merge the two transcripts, verify the compliance to the coding scheme, remove typing and spelling errors, and obtain single transcripts for each participant and each problem, the Perl programming language was used to automate the following operations on the original transcripts:

- Both the videotape and screen capture transcripts were split into two files, each containing the actions for problems 1 and 2 separately. Any problems identified at this point were fixed by hand.

- Using the coding scheme in Table 2 as a syntax definition, each individual action's compliance to the coding syntax was confirmed. If syntax errors were identified, such as misspelled actions, missing information, or mismatched brackets, problems were fixed by hand in order to maintain accurate and valid data.

- As participants would frequently alternate between looking at the screen and looking at the problem description, strict alternation between "RE" and "EX" actions was confirmed. If problems were identified, the original videotape source was checked to correct the error.

- Every action that always leads to a help or search, such as selecting the help or search menu item, was checked for a corresponding "H" or "S" action, in order to ensure that every help and search participants performed would be visible in

the data. If an "H" or "S" was missing, the original screen capture source was used to correct the error.

- Some of the transcripts were coded from the original videotape so many of the initial time stamps were greater or less than time zero (the counters did not start at zero). This discrepancy was used to normalize the transcript to an initial starting time of zero.

- Time stamps were checked to ensure that time was always increasing; any errors were corrected by hand by referencing the videotape or screen captures from which the error originated.

- Quite often, participants would type a portion of a command and finish it minutes later, making the distinction between "C" and "COM" actions important. "COM" actions were not coded manually by the coders, so "C" actions were transformed: ~ in the command string removed a single preceding character and @ represented the execution of a command.

- During experiment sessions, the camcorder and the screen capture software rarely started at the same time. In order to put each transcript on the same time scale, each pair of videotape and screen capture was compared and the time discrepancy estimated in seconds. The videotape and screen capture transcripts were then synchronized and merged together, according to time stamp.

The final data set, after transcripts were created, checked for errors, merged, and split into problems 1 and 2 for each participant, totaling over 33,000 lines of text.

Data Extraction

As manually extracting data from over 33,000 lines of text would be highly prone to errors, not to mention impractical, Perl programs were also written to extract data from the 150 transcripts automatically. A simple language was created to define metrics with, so that variables could be quickly and accurately defined. A parser was also written in Perl in order to interpret these metric definitions and gather data. The grammar for this metric definition language is provided in Figure 19.

METRIC	#(**ACTION**) | ft(**ACTION**) | lt(**ACTION**) | totaldur(**ACTION**) | meandur(**ACTION**) | mindur(**ACTION**) | maxdur(**ACTION**) | distrib(**ACTION**) | first(**ACTION**) | totalchar(**ACTION** *regexp*) | meansize(**ACTION**) | morethanone(**ACTION**)
ACTION	**ATYPE WIN INFO CONTEXT** | seq(**ACTION** { **DISCREP ACTION** }) | =(**ACTION** { **ACTION** }) | pair(**ACTION ACTION**) | cluster(**ACTION**) | byhand(**ACTION**)
ATYPE	*action* | *regexp* | any
WIN	STA | REV | VAR | DO | COM | OUT | DIA | H | any | *regexp*
INFO	**STRING** | **WINOP** | **DIR**
STRING	*regexp* | any
WINOP	MIN | MAX | RE | MV | FR | C
DIR	U | D | L | R
DISCREP	*seconds* | inf
CONTEXT	*regexp* | any

Figure 19. A grammar for the metric definition language used to extract data from transcripts. Bold words are non-terminals, non-bold words are terminals, and italicized words are terminals with special meaning.

The basic premise behind the language was that a metric is a quantification of various properties for a subset of actions in a transcript. For example, to calculate the metric "number of commands executed" for each participant would involve finding the subset of "COM" actions (commands executed), and then calculated the size of the subset. The were numerous ways to identify a subset of the actions in a transcript through the **ACTION** non-terminal:

- Matches a single definition of an action, comprised of the type of action, and the *source*, *info*, and *context* fields. For example, the **ACTION** "WIN H MV

any" would match actions in which the help window was moved, regardless of the **CONTEXT**.

- **seq**: Matches one or more **ACTION**s are provided and separated by a specified number of seconds (or alternatively, an infinite number of seconds). This type of **ACTION** matches sequences of actions separated by specified amounts of time.

- **=**: Matching any action that matches any of the **ACTION**s listed.

- **pair**: Matches pairs of alternating actions, such as those of reading the problem description and examining the computer monitor, irrespective of time between actions.

- **cluster**: Matches all actions matching the provided **ACTION** and clusters together actions that are equivalent; effectively makes a list of distinct actions, removing duplicates. For example, a cluster on help actions would remove all of the duplicate searches for help**.**

- **byhand**: If a metric was too difficult or impossible to define using the language, this type of action could be used to allow each action to be selected on criteria external of the definition.

Each metric was applied to the actions that matched a subset of each participant's transcripts, as defined by the metric. As seen in the grammar, there were numerous types of metrics that could be defined:

- **#**: A simple count of the number of matching actions (i.e., the number of commands executed).

- **ft**, **lt**: The time stamp of the first or last action matched (i.e., the time of the first use of a button).

- **totaldur**, **meandur**, **mindur**, **maxdur**: The total, mean, maximum, or minimum time between a pair of alternating actions (i.e., the total, mean, maximum, or minimum time spent reading the problem description). This metric requires that the action match pairs of actions.

- **distrib**: A non-numerical list of all of the matching actions.

- **first**: The non-numerical literal first matching action.

- **totalchar**: The total number of characters in matching the given regular expression in the *info* field of the action.

- **meansize**: The mean size of a cluster of commands

- **morethanone**: Returns 1 if the size of a cluster of commands is more than 1, and zero otherwise.

The italicized terminals in the grammar have special meanings: *action* represents any action valid name, such as "C" or "WIN," *regexp* represents any valid Perl regular expression, and *seconds* represents a numerical number of seconds. The "any" terminal means to match anything—including nothing—while the "inf" terminal means a limitless number of seconds, used exclusively in the sequence matching action.

With the data extraction system in place, the two coders brainstormed about metrics that might be useful in describing participants' interaction with Stata during the two problems, as well as useful in differentiating those who succeeded and those who seemed to have different strategies. The metrics eventually used are listed in Table 3, Table 4,

and Table 5. These metrics comprised the final set of data that was used to investigate the
research questions posed earlier.

After defining the metrics, each metric was tested to ensure that the definition was
capturing the data intended. As regular expressions are often overly specific, care was
taken to define the metrics to overmatch the data and then specify until the definitions
were gathering exactly what was intended.

Name	Definition	Description
Spoken Word Metrics		
TALKATV	totalword(U none any none /.*/)	Talkativeness
SIGHS	#(U none /.*\[sigh\.*]/ none)	Tendency to sigh
CUSSING	totalword(U none /.*(fuck|shit|damn|crap|dammit|damn it).*/ none /(fuck|shit|damn|crap|dammit|damn it)/)	Tendency to cuss
TIMEREM	#(U none /.*time.*left.*/ none)	Questions about time remaining
FONTCOM	#(U none /.*font.*/ none)	Comments about the font
LAUGHS	#(U none /.*\[laugh\].*/ none)	Tendency to laugh
INQUIS	totalchar(U none /.*\?.*/ none /\?/)	Inquisitiveness
The External Environment		
READPD	#(RE none none none)	References to problem description
READPDT	totaldur(pair(RE none none none EX any none none))	Time spent reading problem description
WRITE	#(WR none any none)	Use of paper to write things down
DRAW	#(DR none any none)	Use of paper to draw pictures
Writing Commands		
TYPOS	totalchar(C none /.*~.*/ none /~/)	Typos
QHATTMP	#(COM none /q$/ none)	Unnecessary attempts to quit help
REPCOM	morethanone(cluster(COM none any none))	Repeated Commands
GUESSYN	#(ERR none /invalid syntax/ none)	Tendency to guess syntax
VARSYN	#(byhand(COM none any none))	Attempts at variations of command syntax
COMDIVR	#(cluster(COM none any none))	Command diversity
COMVOL	#(COM none any none)	Command volume
ERRORS		
HNOTFND	#(seq(ERR none /help for .* not found/ none 15 =(H any any none S any any none)))	Tendency to get help or search after help not found error
UNMAERR	#(ERR none /[\(|\)].*required.*/ none)	Prevalence of unmatched parentheses error
ERRLINK	#(LINK OUT any /.*error)	Use of error description links
DUPERR	morethanone(cluster(ERR none any none))	Prevalence of duplicate error messages
Getting Help		
USENL4H	#(byhand(/(H|S)/ any any none))	Use of natural language for help
FORGETQ	#(Q none none any)	Tendency to use "q" to break
USETUT	#(BAR /(REV|OUT)/ U /.*tutorial.*/)	Use of tutorial examples
HELPTIM	totaldur(pair(=(S any any none H any any none WIN H FR none) =(WIN /(REV|VAR|DO|COM|OUT|DIA)/ FR none C none any none /(INS|DEL)/ DO any any))	Time using help screens
HLPLIST	#(COM none /help list/ none)	Use of "help list" command
USEEXAM	#(CUR /(OUT|H)/ any /.*example.*/)	Use of examples in help files
HELPREL	totaldur(pair(any H any any any /(REV|OUT|VAR|DO|COM|GRA|STA|DIA|ED)/ any any))	Reliance on help window
OUTREL	totaldur(pair(any OUT any any any /(REV|H|VAR|DO|COM|GRA|STA|DIA|ED)/ any any))	Reliance on output window
SBUT	#(BUT H /search/ none)	Use of search button in help window
COMMENU	#(MEN STA /help command/ none)	Use of "Stata command" menu item
SEARCHS	#(S any any none)	Tendency to search
MSRLSPD	meandur(pair(BAR /(H|OUT|DO)/ any any /(WR|RE|DR|EX|WIN|MEN|BUT|CUR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Speed of scrolling in help pages
HELPS	#(H any any none)	Tendency to get help
DUPHELP	morethanone(cluster(H any any none))	Prevalence of duplicate helps
DUPSRCH	morethanone(cluster(S any any none))	Prevalence of duplicate searches

Name	Definition	Description
HELPDIV	#(cluster(H any any none))	Diversity of helps
SRCHDIV	#(cluster(S any any none))	Diversity of searches
Window Management		
WINOFF	#(WIN any MV /.*off.*/)	Tendency to move windows off screen
WINARR	#(WIN any MV any)	Tendency to arrange windows systematically
Use of Graphical User Interface Elements		
INSPBUT	#(CUR /(STA|DO)/ any /.*button.*/)	Inspection of buttons at the top of the Stata window
INSPMEN	#(CUR /(STA|DO)/ any /.*menu.*/)	Inspection of Stata menus
VARWIN	totaldur(pair(any VAR any any any /(REV|OUT|H|DO|COM|GRA|STA|DIA|ED)/ any any))	Use of variables window
REVWIN	totaldur(pair(any REV any any any /(VAR|OUT|H|DO|COM|GRA|STA|DIA|ED)/ any any))	Use of review window
BUTUSE	#(BUT any any none)	Use of buttons
LINKUSE	#(LINK any any any)	Use of links
DUPLINK	morethanone(cluster(LINK any any any))	Links followed multiple times
RETBAR	#(BAR any any /.*back.*/)	Tendency to return scroll bar to original position

Table 3. Metrics defined to extract data from both problems.

Name	Definition	Description
Setting the Number of Objects		
USEQSET	#(COM none /set.*=\s?\d+/ none)	Use of "=" in set commands
USEOBJ	#(COM none /set obj.*/ none)	Tendency to use obj instead of obs in set commands
SETWAST	#(ERR none /obs was 1000, now 1000/ none)	Tendency to set the number of objects unnecessarily
GENOBS	#(COM none /gen ob.*/ none)	Use of generate command to set the number of objects
SET1000	#(COM none /set 1000/ none)	Tendency to type "set 1000"
TUTSET	#(seq(BAR /(REV|OUT)/ U /.*tutorial.*/ 30 COM none /set ob.*/ none))	Use of tutorial examples to write "set obs 1000"
Using Generate		
GENPAUS	#(C none /gen(erate)? .+\s?=\s?$/ none)	Tendency to type "generate varname = " and pause
SET4GEN	#(COM none /set (?!obs).*\s?=\s?/ none)	Use of set command to generate variables
GENUSE	#(COM none /gen / none)	Use of gen instead of generate
Guessing How to do Random		
RNDLTGT	#(COM none /gen(erate)? .+\s?=\s?.*[<>].*/ none)	Use of "<" and ">" to get random numbers
RNDIN	#(COM none /gen(erate)? .+\s?=\s?.* in .*/ none)	Use of the word "in" to get random numbers
RND01	#(COM none /gen(erate)? .+\s?=\s?.*0,\s?1.*/ none)	Use of string "0,1" to get random numbers
GENTO	#(COM none /gen(erate)? .+\s?=\s?.* to .*/ none)	Use of the word "to" to get random numbers
GENCONS	#(COM none /gen(erate)? .+\s?=\s?\d+/ none)	Tendency to generate variables with a uniform value
GENRAND	#(COM none /gen(erate)? .+\s?=\s?.*rand(om)?.*/ none)	Use of the word random to get random numbers
*NL4RND	#(byhand(COM none /.*gen.*/ none))	Use of natural language to get random numbers
Finding the Uniform Function		
USESAMP	#(COM none /sample.*/ none)	Use of "sample" command
GENEXAM	#(seq(H any /generate/ none 120 CUR any any /.*example.*/))	Use of examples in the help for generate
FUNEXAM	#(seq(H any /function(s)?/ none 120 CUR /(H|OUT)/ any /.*example.*/))	Use of examples in the help for functions
USEFUNC	#(H any /function(s)?/ none)	Use of the help on functions
UNIFMLY	#(/(H|S)/ any /uniformly/ none)	Searches for the word "uniformly"
SRNDNUM	#(/(H|S)/ any /.*random.*numbers.*/ none)	Searches on the phrase "random numbers"
FNINTRO	#(seq(H any /function(s)?/ none 120 CUR /(H|OUT)/ any /.*intro.*/))	Use of the introduction heading in the help for functions
HCONTEN	#(H any /contents/ none)	Use of the help contents
Using the Uniform Function		
PARENSP	#(COM none /.*uniform \(.*/ none)	Tendency to ignore parentheses for uniform function
INPAREN	#(COM none /.*uniform\(.+\).*/ none)	Tendency to put something between parentheses in uniform function
INVNORM	#(COM none /.*invnorm.*/ none)	Use of "invnorm()"
UNICNT	#(COM none /.*uniform.*/ none)	Attempts to type "uniform()"
LISTAFT	#(seq(COM none /gen.+=\s?uniform()/ none 30 COM none /list/ none))	Use of "list" command to see observations after successful use
Executing the T Test		
STTST	#(S any /.*t[-]?test.*/ none)	Use of searches to get help on t-test command
PAIRED	#(COM none /ttest .+=.+,\s?unpaired/ none)	Execution of paired t-test
TTSTEXA	#(seq(H any /t[-]?test/ none 30 /(CUR|BAR|SP)/ any any /.*example.*/))	Use of examples in t-test help to create command
TDTEST	#(COM none /.*t-test.*/ none)	Tendency to call t-test "t-test" in commands
TTEST	#(COM none /.*ttest.*/ none)	Tendency to call t-test "ttest" in commands
FRGTCOM	#(COM none /ttest .+=\s?[a-zA-Z0-9]+ unpaired/ none)	Tendency to forget the comma in the t-test command
TTSTUNI	#(seq(COM none /gen(erate)?.*=\s?\d+/ none inf COM none /ttest.*/ none))	Tendency to execute a t-test with uniform data

Name	Definition	Description
Picking a P-value		
SPVALUE	#(S any /.*p-?value.*/ none)	Searches on help for p-value
CHPWHLP	#(seq(COM none /ttest .+=.+,\s?unpaired/ none inf /(H|S)/ any /.*p-?value.*/ none))	Use of help to choose p-value
HAFTTST	#(seq(COM none /ttest .+=.+,\s?unpaired/ none /(H|S)/ any any none))	Help after successful t-test
Strategies		
USECLR	#(COM none /clear/ none)	Use of clear command
REPTUT	#(COM none /gen.*=\s?.+\+.+/ none)	Duplication of tutorial sequence, adding two sets of data
USEED	totaldur(pair(WIN ED FR none =(WIN ED C none WIN /(REV|OUT|VAR|DO|COM|GRA|STA|DIA|H)/ FR none))) Use of the edit or browse window	
TOTREAD	totaldur(pair(RE none none none EX any none none))	Use of the problem description
SETSEED	#(COM none /set seed.*/ none)	Use of the set seed command
Window Management		
MVW2SEE	#(byhand(WIN H MV any))	Do they position the help window so that they can see the command and output windows too?
Performance Variables		
GAVEUP	#(byhand(STOP none /1/ none))	Did they give up?
PGENUNI	#(COM none /gen(erate)? [a-zA-Z0-9]+\s?=\s?uniform\(\)/ none)	Did they generate a uniform variable?
PTTEST	#(COM none /ttest .+\s?=\s?.+,\s?unpaired/ none)	Did they get the t-test to work?
PSETOBS	#(COM none /set obs 1000/ none)	Did they set the number of objects?
TIME	totaldur(pair(START none /1/ none STOP none /1/ none))	Time to complete

Table 4. Metrics defined to extract data from problem 1.

Name	Definition	Description
Comments		
EDITCOM	totalchar(DEL DO any any /./)	Editing of comments
COMTIME	totaldur(pair(CUR DO any /.*comment.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Time spent reading comments
COMOUT	#(INS DO /*/ any)	Tendency to insert comments
TOPCOM	#(BAR DO U /.*top comment.*/)	Use of the comments at the top of the do-file
MODCOMW	#(DEL DO /[a-zA-Z]+/ /.*comment.*/)	Modification of words in comments
MODCOMN	#(DEL DO /\d+/ /.*comment.*/)	Modification of numbers in comments
The Delimit Command		
DELTIME	totaldur(pair(CUR DO any /.*delimit.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Time spent on the delimit command
DELMOD	#(DEL DO any /.*delimit.*/)	Tendency to modify the delimit command
The For Loop		
FORTIME	totaldur(pair(CUR DO any /.*(for|loop).*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Time spent on the for loop
LPLINE	#(byhand(INS DO any /.*(for|loop).*/))	Tendency to put for loop commands on individual lines
REPLNUM	#(seq(DEL DO /num/ any 10 INS DO /X/ any))	Was "num" changed to "X"?
REPLX	#(seq(DEL DO /X/ any 10 INS DO /num/ any))	Was "X" changed to "num"?
ROFPHLP	#(/(S|H)/ any /(r\(p\)|r|p)$/ none)	Tendency to search for help on "r(p)" or "p" or "r"?
X2Y	#(seq(DEL DO /X/ any 10 INS DO /Y/ any))	Was "X" changed to "Y" in the for loop?
CHNFOR5	#(DEL DO /5/ /.*for.*/)	Tendency to change the "5" in "replace rand1 = X in 1/5"
Generate Rand Commands		
GRNDTIM	totaldur(pair(CUR DO any /.*gen.*rand.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Time spent on the commands generating the two data sets
CONFGRN	#(seq(CUR DO any /.*gen.*rand.*/ 30 BAR REV U /.*tutorial.*/))	Were the generate rand commands confirmed?
The Set Obs Command		
SETTIME	totaldur(pair(CUR DO any /.*set obs.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|CUT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	How much time was spent on the set obs command?
CONFSET	#(seq(CUR DO any /.*set obs.*/ 30 =(BAR REV U /.*tutorial.*/ CUR OUT any /.*set obs.*/)))	Was the set command confirmed?
The Generate P-Values Command		
GNPTIME	totaldur(pair(CUR DO any /.*gen.*p-?val.*/	Time spent on the gen pvalues command

	/(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|C UT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	
GNP210	#(byhand(/(DEL|INS)/ DO any /.*gen.*p-?val.*/))	Was 10 inserted into the gen pvalues command?
GNP2TH	#(byhand(/(DEL|INS)/ DO any /.*gen.*p-?val.*/))	Was 1000 inserted into the gen pvalues command?
GNPCH	#(byhand(/(DEL|INS)/ DO any /.*gen.*p-?val.*/))	Tendency to change the 100 in 1/100

The Generate Value of Outlier Command

GVLTIME	totaldur(pair(CUR DO any /.*gen val.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|C UT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	Time spent on the generate valueofoutlier command
REMUNDR	#(DEL DO /_/ /.*gen val.*/)	Was the underscore removed?
GVL210	#(byhand(/(DEL|INS)/ DO any /.*gen(erate)? val.*/))	Was 10 inserted into the gen valueofoutlier command?
GVL2TH	#(byhand(/(DEL|INS)/ DO any /.*gen(erate)? val.*/))	Was 1000 inserted into the gen valueofoutlier command??
GVLCH	#(byhand(/(DEL|INS)/ DO any /.*gen(erate)? val.*/))	Tendency to change the 100 in 1/100

The Graph Command

GRPTIME	totaldur(pair(CUR DO any /.*graph com.*/ /(WR|RE|DR|EX|WIN|MEN|BUT|BAR|LINK|INS|DEL|SEL|COP|PAS|C UT|H|S|SP|Q|RUN|C|ERR|COM)/ any any any))	How much time was spent on the graph command?
GRPTIT	#(DEL DO any /.*graph.*title.*/)	Was the titled changed?
GRPCOM	#(DEL DO any /.*graph.*comment.*/)	Was the graph comment changed?

Running the Do-File

DOUSE	#(BUT DO /do/ none)	Use of the do button
RUNUSE	#(RUN none none none)	Use of the run button
IGNERR	#(seq(RUN none none none 30 ERR none any none))	Tendency to see errors soon after running
ERRS	#(ERR none any none)	Tendency to see errors
IMPAT	#(seq(RUN none none none 7 RUN none none none))	Tendency to run repeatedly, impatiently

The Graph

GRATIME	totaldur(pair(WIN GRA FR none =(WIN GRA C none WIN DO FR none)))	Time spent viewing the graph
T2VGRP	ft(WIN GRA FR none)	Eagerness to view the graph
SIZEGRA	#(WIN GRA RE any)	Tendency to resize the graph window
TRACEGR	#(CUR GRA any /.*trace.*/)	Tendency to trace the curve of the graph

Strategies

T2VDO	ft(WIN DO FR none)	Eagerness to edit the file
FIXSEMI	#(INS DO /;/ any)	Tendency to correct semi colons
SAVEDO	#(BUT DO /save/ none)	Tendency to save the file
REPCHNG	morethanone(cluster(INS DO any any))	Tendency to repeat the same changes
CHNUM	#(DEL DO /\d+/ any)	Tendency to change numbers
CHWORD	#(DEL DO /[A-Za-z]+/ any)	Tendency to change words
EXMAN	#(COM none /(set|gen|ttest|for|graph|delimit)/ none)	Tendency to execute commands manually from Stata
NCHB4R	#(seq(INS DO any any inf INS DO any any inf RUN none none none))	Number of times >=2 changes made before run
TINDO	totaldur(pair(WIN DO FR none WIN /(ED|REV|OUT|VAR|COM|GRA|STA|DIA|H)/ FR none))	Time in do file
LOOKDAT	#(=(COM none /list/ none WIN ED FR none))	Tendency to inspect data with "list" or edit window?
TRYCLR	#(COM none /clear/ none)	Tendency to clear data set

Reading the File

SEELEN	#(seq(START none /2/ none inf WIN DO FR none 60 BAR DO D /.*very.*bottom.*/))	Do they go to the bottom real fast to see how long it is?

Getting Help

HLPGR	#(/(H|S)/ any /.*graph.*/ none)	Did they search for help on the graph command?
HLPFOR	#(/(H|S)/ any /.*for.*/ none)	Did they search for help on the for command?
HLPGEN	#(/(H|S)/ any /.*generate.*/ none)	Did they search for help on the generate command?
HLPTTST	#(/(H|S)/ any /.*t[-]?test.*/ none)	Did they search for help on the ttest command?
HLPDEL	#(/(H|S)/ any /.*delimit.*/ none)	Did they search for help on the delimit command?
HLPSLSH	#(/(H|S)/ any /[\\\/]/ none)	Did they search for help on / or \?
HLPDO	#(/(H|S)/ any /.*do.*/ none)	Did they search for help on do files?

Performance Variables

PAXES	#(byhand(STOP none /2/ none))	Did they fix the axes?
PFOR	#(byhand(STOP none /2/ none))	Did they fix the for-loop bug?
PGNPVL	byhand(STOP none /2/ none))	Did they fix the range on the "generate pvalues" bug?
PGNVAL	#(byhand(STOP none /2/ none))	Did they fix the range on the "generate valueofoutlier" bug?
TIME	totaldur(pair(START none /2/ none STOP none /2/ none))	Time to complete

Table 5. Metrics defined to extract data from problem 2.

Results

There were two areas of participants' interactions that this experiment intended to explore: the influence of individual differences on success, and whether or not groups of programming, testing, and debugging style would naturally cluster together and provide predictive value. Overall performance on the programming and testing and debugging problem is considered first.

Overall Performance

Participants overall performance on problems 1 and 2 is provided in Table 6. Nearly every participant succeeded in creating the data set for problem 1, but only half of the participants created the two variables of uniformly distributed numbers and executed the t test to compare the two variables. In problem 2, we can see that over two thirds of the participants found the bug in the "generate valueofoutlier" command (which had a comment immediately above it that basically identified the bug). Only a third of the participants found the nearly identical bug in the "generate pvalues" command, which did not have a comment identifying the bug. A third of the participants also changed the for loop iteration range, despite its lack of impact on the data or the graph. Finally, only about 20 percent of the participants noticed the inverted axes in the graph.

Milestone	Success	Failure	% Success
Problem 1			
Creating a data set of 1000 objects	70	5	93.3
Creating two sets of uniformly distributed random numbers	38	37	50.7
Executing a t test to compare the two data sets	34	41	45.3
Problem 2			
Correcting the inverted axes in the graph	16	59	21.3
Changing the for loop iteration range from "10/1" to "1/10"	25	50	33.3
Changing the observation range in the generate pvalues command to "1/10"	28	47	37.3
Changing the observation range in the generate valueofoutlier command to "1/10"	47	28	62.7

Table 6. Frequency of success, failure, and percent of participants succeeding on each milestone in problems 1 and 2.

Tabulated detail about problem 1 milestones is presented in Table 7 reveals more detail about the nature of the performance on the problem. Most individuals did one of the following:

- Succeeded only at generating the data set of 1000 objects (29)

- Succeeded at the whole problem (28)

- Succeeded at generating the data set of 1000 objects and creating the two sets of uniformly distributed random numbers (8)

A small number executed the t test without the random numbers; a smaller number failed at all tasks. In general, participants solved the problem in sequence and did not work out of sequence.

Generated the Data Set of 1000 Objects	Created Two Sets of Uniformly Distributed Random Numbers	Executed the T Test	
		Failure	Success
Failure	Failure	3	0
	Success	1	1
Success	Failure	29	5
	Success	8	28

Table 7. Tabulations of success on the three milestones in problem 1.

Similar data is provided for problem 2 by each bug in Table 8. Most individuals did one of the following:

- Did not fix any bugs (14)

- Fixed the observation range in the *generate valueofoutlier* range (10)

- Fixed the observation range in the *generate valueofoutlier* and *generate pvalues* range (10)

- Fixed only the for loop iteration range (7)

- Fixed the *generate valueofoutlier* range and the for loop iteration range (7)

- Fixed the *generate pvalue* range and the for loop iteration range (6)

The few participants who fixed the inverted axes tended to also fix the observation range in the *generate valueofoutlier* range but failed to fix the for loop iteration range.

Changing the observation range in the *generate valueofoutlier* command	Changing the Observation Range in the *generate pvalues* command	Correcting the Inverted Axes in the Graph	Changing the For Loop Iteration Range	
			Failure	Success
Failure	Failure	Failure	14	7
		Success	1	0
	Success	Failure	4	1
		Success	1	0
Success	Failure	Failure	10	7
		Success	5	3
	Success	Failure	10	6
		Success	5	1

Table 8. Tabulations of success on the four bugs in problem 2.

Individual Differences and Performance

The performance data provided in Table 6 hides the individual differences behind successful completion of the problems. For the following analyses various individual differences were compared with performances on each problem. Initial, coarse analyses are presented in Table 9. Measures of intelligence (VCog I and Vocab27), experience with statistics, mathematics, programming, and computer use (from the statistics test and

the background questionnaire), and attitudes towards mathematics, computers, statistics, and the experiment itself (from the background questionnaire), were correlated with overall performance on problems 1 and 2. Overall performance was calculated by taking a mean of the dichotomous performance variables.

For problem 1, higher intelligence, more experience with mathematics, computer programming, and computers, as well as more positive attitudes towards computers and mathematics were all significantly correlated with success on the first problem. It should be noted that performance on the statistics test as well as self-reported statistics experience showed absolutely no relationship, suggesting that the background knowledge supposedly necessary for completing the task (which we have seen was in general poor), was unnecessary. For problem 2, more experience with mathematics, and more positive attitudes towards mathematics and computers were associated with higher overall performance. Intelligence, programming experience, and statistical knowledge had little relationship to success.

Characteristic	Problem 1 Performance	Problem 2 Performance
VCog I	**.452**	.143
Vocab27	**.318**	.034
Statistics Test	-.058	.016
Self-reported math abilities	**.510**	**.352**
Self-reported statistics abilities	-.062	-.011
Self-reported programming experience	**.522**	.202
Self-reported computer experience	**.301**	.116
Attitudes toward the experiment	-.174	.038
Attitudes towards computers	**.617**	**.272**
Attitudes towards statistics	.211	.189
Attitudes towards mathematics	**.443**	**.274**

Table 9. Correlations between measures of intelligence, experience, and attitudes with overall performance on both problems. Correlations significant at an a of .05 are highlighted in bold.

In order to unearth the true relationships between success and major, gender, intelligence, experience, and attitudes, multivariate ANOVAs for each measure of

intelligence, experience, and attitudes were performed by major, gender, and each milestone for problem 1 and 2.Intelligence, experience, and attitudes will be considered in turn, each with major, gender, and each of the performance milestones for problems 1 and 2.

Intelligence, Major, Gender, and Performance

Measures of verbal IQ, general problem solving intelligence, and statistics competency are presented by major, gender, and milestone in Table 10. At this level of detail, it is clear that there were few strong predictors of success. There was a significant interaction effect between major and creating the data set of 1000 objects for the Vocab27 test, showing that the computer science majors who did not succeed at this milestone tended to have higher verbal intelligence. However, this data is unlikely to have any validity given the small number of individuals who failed at this task. Main effects for creating the random numbers were found for the Vocab27 and VCog I test, showing that more intelligent participants succeeded. A significant interaction effect between major and executing the t test was found for VCog I, showing that psychology and other majors who succeeded were more intelligent, while computer science majors who succeeded were less intelligent. A significant interaction effect between gender and executing the t test was found for the statistics test, suggesting that females who failed at executing the t test were slightly less competent in statistics. Finally, a main effect on the statistics test was also found for executing the t test, showing that those failing at executing the t test were slight more competent in statistics.

			Created Data Set		Created Random Numbers		Executed T Test		Fixed Axes		Fixed *gen pvalues*		Fixed *gen valueof...*		Fixed Loop Range	
	Maj.	**Sex**	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Vocab27	Psy	m	0.00	0.57	0.52	0.65	0.52	0.70	0.57	0.00	0.58	0.54	0.58	0.56	0.56	0.58
		f	0.00	0.51	0.51	0.00	0.52	0.37	0.50	0.63	0.50	0.53	0.56	0.44	0.51	0.00
		Tot.	0.00	0.53	0.51	0.65	0.52	0.59	0.53	0.63	0.53	0.53	0.56	0.50	0.53	0.58
	CS	m	0.85	0.52	0.41	0.57	0.62	0.52	0.56	0.47	0.51	0.58	0.57	0.52	0.54	0.53
		f	0.00	0.68	0.00	0.68	0.00	0.68	0.68	0.00	0.78	0.63	0.78	0.63	0.81	0.61
		Tot.	0.85	0.54	0.41	0.58	0.62	0.54	0.58	0.47	0.53	0.59	0.60	0.53	0.57	0.54
	O	m	0.49	0.47	0.39	0.56	0.44	0.56	0.42	0.56	0.50	0.44	0.43	0.49	0.46	0.57
		f	0.37	0.53	0.46	0.59	0.47	0.63	0.52	0.55	0.53	0.51	0.49	0.55	0.50	0.56
		Tot.	0.46	0.51	0.44	0.58	0.46	0.60	0.48	0.55	0.52	0.48	0.47	0.52	0.48	0.56
	Tot.	m	0.58	0.52	0.44	0.58	0.50	0.54	0.53	0.51	0.52	0.53	0.54	0.52	0.51	0.55
		f	0.37	0.54	0.49	0.61	0.50	0.62	0.53	0.56	0.53	0.53	0.54	0.53	0.52	0.57
		Tot.	0.54	0.53	0.47	0.59	0.50	0.57	0.53	0.53	0.53	0.53	0.54	0.52	0.51	0.56
VCog I	PSY	m	0.00	0.59	0.54	0.68	0.57	0.65	0.59	0.00	0.58	0.62	0.57	0.60	0.57	0.63
		f	0.00	0.57	0.57	0.00	0.57	0.50	0.56	0.64	0.57	0.56	0.61	0.51	0.57	0.00
		Tot.	0.00	0.58	0.56	0.68	0.57	0.60	0.57	0.64	0.58	0.58	0.60	0.56	0.57	0.63
	CS	m	0.72	0.61	0.57	0.63	0.69	0.61	0.64	0.57	0.60	0.65	0.63	0.62	0.62	0.61
		f	0.00	0.68	0.00	0.68	0.00	0.68	0.68	0.00	0.70	0.67	0.70	0.67	0.88	0.58
		Tot.	0.72	0.62	0.57	0.64	0.69	0.62	0.65	0.57	0.61	0.65	0.64	0.62	0.65	0.61
	O	m	0.47	0.64	0.49	0.71	0.54	0.73	0.55	0.68	0.60	0.60	0.55	0.62	0.58	0.69
		f	0.56	0.59	0.52	0.65	0.54	0.68	0.57	0.63	0.59	0.58	0.57	0.59	0.57	0.61
		Tot.	0.50	0.60	0.51	0.68	0.54	0.70	0.56	0.65	0.59	0.59	0.56	0.61	0.57	0.63
	Tot.	m	0.54	0.62	0.53	0.66	0.58	0.63	0.60	0.62	0.60	0.63	0.59	0.61	0.60	0.63
		f	0.56	0.59	0.55	0.66	0.56	0.66	0.58	0.63	0.59	0.59	0.60	0.58	0.58	0.61
		Tot.	0.54	0.60	0.54	0.66	0.56	0.64	0.59	0.62	0.59	0.61	0.59	0.60	0.59	0.62
Statistics	PSY	m	0.00	0.25	0.22	0.30	0.25	0.25	0.25	0.00	0.25	0.25	0.13	0.32	0.32	0.13
		f	0.00	0.33	0.33	0.00	0.35	0.10	0.29	0.70	0.34	0.30	0.40	0.22	0.33	0.00
		Tot.	0.00	0.30	0.29	0.30	0.31	0.20	0.27	0.70	0.30	0.28	0.32	0.27	0.32	0.13
	CS	m	0.50	0.35	0.40	0.35	0.53	0.33	0.32	0.45	0.33	0.40	0.43	0.33	0.32	0.39
		f	0.00	0.27	0.00	0.27	0.00	0.27	0.27	0.00	0.40	0.20	0.40	0.20	0.30	0.25
		Tot.	0.50	0.34	0.40	0.34	0.53	0.32	0.31	0.45	0.34	0.36	0.43	0.31	0.32	0.37
	O	m	0.27	0.37	0.30	0.38	0.30	0.43	0.34	0.34	0.36	0.32	0.37	0.33	0.35	0.30
		f	0.60	0.32	0.35	0.32	0.38	0.25	0.34	0.33	0.38	0.27	0.29	0.37	0.34	0.33
		Tot.	0.35	0.34	0.33	0.35	0.35	0.32	0.34	0.33	0.37	0.29	0.31	0.36	0.35	0.32
	Tot.	m	0.33	0.33	0.30	0.35	0.32	0.34	0.31	0.40	0.32	0.35	0.34	0.33	0.33	0.33
		f	0.60	0.32	0.34	0.31	0.36	0.24	0.31	0.40	0.36	0.27	0.34	0.31	0.33	0.31
		Tot.	0.38	0.33	0.32	0.34	0.35	0.31	0.31	0.40	0.34	0.31	0.34	0.32	0.33	0.32

Table 10. Mean scores on tests of intelligence and statistics by gender, major, and success on milestones of problems 1 and 2. Dark cells represent a main effect, medium-colored cells an interaction effect of gender or major by performance, and the lightest cells an interaction effect between major, gender, and performance (alpha less than .01).CS = Computer Science, PSY = Psychology, and O = Other.

With regard to problem 2, there were few significant individual difference predictors of success. There was an interaction effect between major and fixing the inverted axes for the statistics test, showing that computer science and psychology majors who succeeded were much more competent in statistics whereas there was no such difference in other majors. Finally, there was a main effect of fixing the inverted axes for the statistics test showing that participants who were successful at this task had higher statistics competency. It should also be noted that the insignificant results are also interesting. For example, statistics competency and intelligence seemed to have no influence on success at fixing the bugs in problem 2.

Self-Reported Experience, Major, Gender, and Performance

Similar results are provided for self-reported experience by major, gender, and performance in Table 11. There was a main interaction effect for creating the data set and major; computer scientists who failed at the task had slightly less programming experience whereas other majors who failed at the task had slightly higher programming experience. There was an interaction effect of mathematics experience for creating random numbers and major; psychology and other majors who succeeded at the task had more math experience but computer science majors who succeeded tended to have less mathematics experience. There was also a significant interaction effect between creating random numbers and gender for math experience; in general, males and females tended to have higher math experience if successful on creating the random numbers. Though there was a visible different overall on math experience and creating random numbers, it was not significant. There was a main interaction effect on programming experience and

creating the random numbers: participants successful at this milestone had more programming experience. Finally, there was a significant main effect for executing the t test and statistics software experience; those who were successful had slightly less experience.

			Self-Reported Experience (0 = none, 1 = least, 7 = most)												
			Created Data Set		Created Random Numbers		Executed T Test		Fixed Axes		Fixed *gen pvalues*		Fixed *gen valueof...*		Fixed Loop Range	
Maj.	Sex		No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Math	PSY	m	0.00	3.91	3.35	4.83	3.83	4.13	3.91	0.00	3.79	4.25.	2.67	4.65	4.30	3.25
		f	0.00	2.96	2.96	0.00	2.80	4.75	3.00	2.50	2.94	3.00	2.93	3.00	2.96	0.00
		Tot.	0.00	3.34	3.07	4.83	3.16	4.33	3.38	2.50	3.30	3.42	2.85	3.83	3.35	3.25
	CS	m	5.75	5.54	5.88	5.47	5.58	5.54	5.47	5.75	5.44	5.72	5.50	5.57	5.78	5.34
		f	0.00	5.25	0.00	5.25	0.00	5.25	5.25	0.00	4.75	5.50	4.75	5.50	5.00	5.38
		Tot.	5.75	5.50	5.88	5.44	5.58	5.50	5.43	5.75	5.39	5.68	5.39	5.56	5.70	5.35
	O	m	4.08	5.25	3.33	6.58	4.06	6.75	4.11	6.15	4.82	5.15	3.25	5.53	4.70	6.25
		f	6.75	4.60	4.25	5.22	4.44	5.29	4.63	5.00	5.17	3.93	4.19	5.09	4.50	5.07
		Tot.	4.75	4.81	3.91	5.77	4.30	5.88	4.47	5.64	5.04	4.44	3.93	5.29	4.59	5.33
	Tot.	m	4.50	5.11	4.02	5.65	4.25	5.63	4.73	5.93	4.89	5.33	4.23	5.40	5.05	5.06
		f	6.75	4.06	3.55	5.23	3.69	5.23	4.08	4.50	4.30	3.88	3.67	4.56	3.78	5.14
		Tot.	4.95	4.62	3.74	5.52	3.92	5.51	4.41	5.48	4.63	4.66	3.91	5.07	4.42	5.09
Statistics	PSY	m	0.00	1.25	0.75	2.08	1.17	1.50	1.25	0.00	1.00	2.00	0.92	1.45	1.80	0.33
		f	0.00	1.94	1.94	0.00	1.91	2.25	1.93	2.00	2.22	1.38	2.00	1.85	1.94	0.00
		Tot.	0.00	1.66	1.59	2.08	1.65	1.75	1.64	2.00	1.70	1.58	1.68	1.65	1.90	0.33
	CS	m	0.00	1.14	1.13	1.07	1.00	1.10	0.88	1.58	1.21	0.88	1.13	1.07	1.28	0.91
		f	0.00	2.25	0.00	2.25	0.00	2.25	2.25	0.00	3.00	1.88	3.00	1.88	1.50	2.63
		Tot.	0.00	1.28	1.13	1.25	1.00	1.26	1.11	1.58	1.34	1.08	1.39	1.16	1.30	1.17
	O	m	1.92	2.22	2.17	2.13	2.22	2.00	2.11	2.20	2.14	2.15	2.08	2.17	2.10	2.38
		f	0.00	1.33	1.50	1.00	1.21	1.38	1.15	1.69	0.90	1.89	1.25	1.27	1.27	1.25
		Tot.	1.44	1.63	1.75	1.45	1.60	1.63	1.45	1.97	1.36	2.00	1.48	1.68	1.65	1.50
	Tot.	m	1.44	1.43	1.42	1.43	1.63	1.28	1.27	1.86	1.41	1.45	1.31	1.47	1.71	0.98
		f	0.00	1.64	1.74	1.31	1.53	1.73	1.56	1.75	1.50	1.73	1.69	1.50	1.60	1.56
		Tot.	1.15	1.53	1.61	1.39	1.57	1.41	1.41	1.83	1.45	1.58	1.53	1.48	1.66	1.19
Programming	PSY	m	0.00	0.58	0.67	0.44	0.56	0.67	0.58	0.00	0.78	0.00	0.00	0.93	0.67	0.44
		f	0.00	0.31	0.31	0.00	0.33	0.00	0.33	0.00	0.40	0.13	0.52	0.00	0.31	0.00
		Tot.	0.00	0.42	0.41	0.44	0.41	0.44	0.44	0.00	0.56	0.08	0.37	0.47	0.41	0.44
	CS	m	3.17	4.29	4.25	4.24	5.06	4.10	4.07	4.67	4.49	3.83	4.31	4.21	4.42	4.08
		f	0.00	3.39	0.00	3.39	0.00	3.39	3.39	0.00	2.33	3.92	2.33	3.92	3.83	3.17
		Tot.	3.17	4.17	4.25	4.11	5.06	4.00	3.95	4.67	4.33	3.85	4.02	4.18	4.36	3.94
	O	m	1.56	2.41	1.83	2.56	2.00	2.58	2.10	2.33	2.88	1.23	3.56	1.74	2.13	2.50
		f	3.00	0.84	0.30	1.69	0.60	1.72	0.99	0.83	0.92	1.02	0.83	1.05	0.67	1.45
		Tot.	1.92	1.36	0.88	2.03	1.13	2.07	1.34	1.67	1.64	1.11	1.58	1.36	1.33	1.69
	Tot.	m	1.96	3.03	2.09	3.41	2.03	3.56	2.68	3.61	3.20	2.46	3.04	2.88	2.75	3.20
		f	3.00	0.88	0.30	2.11	0.48	2.05	0.99	0.67	0.79	1.19	0.79	1.07	0.62	1.83
		Tot.	2.17	2.02	1.03	3.00	1.12	3.12	1.85	2.69	2.12	1.87	1.76	2.19	1.69	2.71

Computers	PSY	m	0.00	5.44	5.20	5.83	5.33	5.75	5.44	0.00	5.67	4.75	4.50	6.00	5.60	5.17
		f	0.00	5.50	5.50	0.00	5.55	5.00	5.41	6.50	5.56	5.38	5.71	5.20	5.50	0.00
		Tot.	0.00	5.48	5.41	5.83	5.47	5.50	5.42	6.50	5.61	5.17	5.35	5.60	5.53	5.17
	CS	m	6.50	6.10	6.25	6.09	5.33	6.25	6.13	6.08	5.96	6.38	6.08	6.13	6.15	6.09
		f	0.00	5.83	0.00	5.83	0.00	5.83	5.83	0.00	7.00	5.25	7.00	5.25	6.00	5.75
		Tot.	6.50	6.07	6.25	6.05	5.33	6.19	6.08	6.08	6.04	6.15	6.21	6.03	6.14	6.04
	O	m	6.00	6.06	5.92	6.17	5.81	6.50	6.00	6.10	6.21	5.80	6.33	5.94	6.15	5.50
		f	6.00	5.64	5.15	6.22	5.42	6.17	5.57	6.00	5.63	5.71	5.25	5.95	5.54	5.86
		Tot.	6.00	5.78	5.44	6.20	5.57	6.30	5.70	6.06	5.84	5.75	5.55	5.95	5.82	5.78
	Tot.	m	6.13	5.95	5.77	6.08	5.56	6.25	5.92	6.09	5.96	5.97	5.75	6.05	6.04	5.84
		f	6.00	5.61	5.34	6.13	5.48	5.95	5.53	6.10	5.67	5.54	5.56	5.67	5.54	5.83
		Tot.	6.10	5.79	5.51	6.09	5.51	6.16	5.73	6.09	5.83	5.77	5.64	5.90	5.79	5.84

Table 11. Mean self-reported responses on experience with math, computers, statistics software, and programming by gender, major, and success on milestones of problems 1 and 2. Dark cells represent a main effect, medium-colored cells an interaction effect of gender or major by performance, and the lightest cells an interaction effect between major, gender, and performance (alpha less than .01).CS = Computer Science, PSY = Psychology, and O = Other.

Again, there were few significant individual difference predictors of success at fixing the bugs in problem 2. There was a main effect for math experience on fixing the *generate valueofoutlier* bug; successful participants had more mathematics experience. There was also a significant interaction effect between gender and the *generate pvalues* bug; males who had less programming experience were more successful at this bug while females who had more experience were more successful. Again, some of the negative findings are important: computer, programming, and statistics software experience had no relationship to success on fixing problem 2 bugs.

Attitudes, Major, Gender, and Performance

Finally, results are provided for attitudes by major, gender, and performance in Table 12. There was a significant interaction effect between major and the creation of the data set of 1000 objects; other majors who failed at the task had slightly higher opinions of computers. There was also a main effect on creating the random numbers; participants

successful at this task had more positive attitudes about computers. Also for the create

random numbers milestone, there was a significant interaction effect with attitudes

towards mathematics: psychology and other majors who succeeded had slightly more

positive attitudes of mathematics while computer science majors who succeeded had

slightly more negative attitudes towards mathematics.

			Attitudes (1 = negative, 7 = positive)												
			Created Data Set		Created Random Numbers		Executed T Test		Fixed Axes		Fixed *gen pvalues*		Fixed *gen valueof...*		Fixed Loop Range	
Maj.	**Sex**		No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes	No	Yes
Experiment PSY	m		0.00	4.74	4.55	5.00	4.63	5.00	4.72	0.00	4.79	4.50	4.00	5.15	4.95	4.33
	f		0.00	4.52	4.52	0.00	4.52	4.50	4.52	4.50	4.81	3.94	4.64	4.35	4.52	0.00
	Tot.		0.00	4.60	4.53	5.00	4.56	4.83	4.61	4.50	4.80	4.13	4.45	4.75	4.65	4.33
CS	m		3.75	5.44	5.44	5.34	5.67	5.31	5.40	5.25	5.46	5.19	5.25	5.40	5.38	5.34
	f		0.00	4.42	0.00	4.42	0.00	4.42	4.42	0.00	3.75	4.75	3.75	4.75	5.00	4.13
	Tot.		3.75	5.30	5.44	5.20	5.67	5.18	5.24	5.25	5.34	5.10	5.04	5.32	5.34	5.15
O	m		4.33	5.06	4.46	5.29	4.50	5.63	4.71	5.10	4.89	4.85	4.75	4.92	4.93	4.63
	f		3.50	4.90	4.58	5.11	4.62	5.29	4.68	5.38	4.71	5.04	4.88	4.80	4.65	5.14
	Tot.		4.13	4.95	4.53	5.18	4.57	5.43	4.69	5.22	4.78	4.96	4.84	4.85	4.77	5.03
Tot.	m		4.19	5.19	4.75	5.29	4.75	5.33	5.06	5.18	5.15	4.98	4.81	5.21	5.11	5.06
	f		3.50	4.72	4.55	4.94	4.57	4.95	4.59	5.20	4.70	4.65	4.70	4.67	4.60	4.92
	Tot.		4.05	4.97	4.63	5.18	4.65	5.22	4.83	5.19	4.95	4.83	4.75	5.00	4.86	5.01
Computers PSY	m		0.00	4.28	3.90	4.92	3.83	5.63	4.28	0.00	4.75	2.88	3.58	4.70	4.40	4.08
	f		0.00	3.88	3.88	0.00	3.93	3.25	3.86	4.00	4.38	2.88	4.11	3.55	3.88	0.00
	Tot.		0.00	4.04	3.88	4.92	3.90	4.83	4.04	4.00	4.54	2.88	3.95	4.13	4.03	4.08
CS	m		6.50	6.45	6.38	6.47	6.50	6.44	6.43	6.50	6.48	6.41	6.63	6.38	6.60	6.32
	f		0.00	6.17	0.00	6.17	0.00	6.17	6.17	0.00	5.50	6.50	5.50	6.50	6.25	6.13
	Tot.		6.50	6.41	6.38	6.43	6.50	6.40	6.39	6.50	6.41	6.43	6.46	6.40	6.57	6.29
O	m		5.17	5.58	4.92	6.04	5.06	6.31	5.29	5.75	5.57	5.35	5.33	5.53	5.48	5.50
	f		6.00	4.82	4.20	5.64	4.54	5.63	4.75	5.38	4.73	5.14	4.28	5.32	4.77	5.07
	Tot.		5.38	5.07	4.47	5.80	4.74	5.90	4.92	5.58	5.04	5.23	4.57	5.41	5.09	5.17
Tot.	m		5.50	5.77	4.97	6.19	4.88	6.35	5.59	6.16	5.84	5.58	5.54	5.83	5.71	5.80
	f		6.00	4.60	4.02	5.77	4.26	5.55	4.56	5.10	4.63	4.65	4.28	4.96	4.40	5.31
	Tot.		5.60	5.22	4.41	6.06	4.52	6.12	5.08	5.83	5.30	5.15	4.82	5.49	5.06	5.62
Statistics PSY	m		0.00	2.81	2.10	4.00	2.38	4.13	2.81	0.00	2.83	2.75	2.17	3.20	2.80	2.83
	f		0.00	2.77	2.77	0.00	2.59	4.75	2.80	2.50	3.19	1.94	2.89	2.60	2.77	0.00
	Tot.		0.00	2.79	2.57	4.00	2.51	4.33	2.80	2.50	3.04	2.21	2.68	2.90	2.78	2.83
CS	m		3.25	3.44	3.63	3.38	2.92	3.51	3.48	3.29	3.31	3.63	3.25	3.50	3.75	3.14
	f		0.00	3.83	0.00	3.83	0.00	3.83	3.83	0.00	3.75	3.88	3.75	3.88	3.00	4.25
	Tot.		3.25	3.49	3.63	3.45	2.92	3.56	3.54	3.29	3.34	3.68	3.32	3.54	3.68	3.31
O	m		3.67	4.08	3.92	4.04	3.94	4.06	3.82	4.20	3.82	4.20	3.17	4.25	3.98	4.00
	f		3.50	3.31	3.33	3.31	3.29	3.38	3.13	4.00	2.94	3.96	2.88	3.64	3.65	2.75
	Tot.		3.63	3.56	3.55	3.60	3.54	3.65	3.35	4.11	3.26	4.06	2.95	3.91	3.80	3.03
Tot.	m		3.56	3.46	3.23	3.61	3.21	3.66	3.38	3.70	3.34	3.70	2.96	3.68	3.65	3.19

		f	3.50	3.16	3.02	3.44	2.97	3.65	3.08	3.70	3.07	3.33	2.94	3.38	3.20	3.08
		Tot.	3.55	3.32	3.11	3.55	3.07	3.65	3.23	3.70	3.22	3.53	2.95	3.56	3.43	3.15
Math	PSY	m	0.00	3.63	2.80	5.00	3.25	4.75	3.63	0.00	3.63	3.63	2.75	4.15	4.25	2.58
		f	0.00	3.67	3.67	0.00	3.52	5.25	3.86	1.50	3.88	3.25	3.96	3.25	3.67	0.00
		Tot.	0.00	3.65	3.41	5.00	3.43	4.92	3.76	1.50	3.77	3.38	3.60	3.70	3.84	2.58
	CS	m	5.25	5.36	5.94	5.22	5.83	5.28	5.18	5.79	5.21	5.59	5.25	5.40	5.53	5.20
		f	0.00	5.25	0.00	5.25	0.00	5.25	5.25	0.00	4.25	5.75	4.25	5.75	5.00	5.38
		Tot.	5.25	5.35	5.94	5.23	5.83	5.27	5.19	5.79	5.14	5.63	5.11	5.44	5.48	5.23
	O	m	3.58	5.61	4.13	6.08	4.63	6.06	4.50	5.95	4.89	5.40	3.67	5.58	4.90	6.13
		f	6.75	4.31	3.65	5.31	4.13	5.08	4.33	4.81	4.65	4.07	3.72	4.95	4.42	4.46
		Tot.	4.38	4.74	3.83	5.62	4.32	5.48	4.39	5.44	4.74	4.63	3.70	5.24	4.64	4.83
	Tot.	m	4.00	5.05	4.17	5.39	4.35	5.36	4.61	5.86	4.76	5.27	4.23	5.24	5.02	4.83
		f	6.75	4.16	3.66	5.29	3.85	5.15	4.25	4.15	4.33	4.08	3.86	4.57	4.08	4.67
		Tot.	4.55	4.63	3.86	5.36	4.06	5.30	4.43	5.33	4.57	4.71	4.02	4.98	4.55	4.77

Table 12. Mean responses on attitudes towards math, programming, statistics, and the experiment by gender, major, and success on milestones of problems 1 and 2. Dark cells represent a main effect, medium-colored cells an interaction effect of gender or major by performance, and the lightest cells an interaction effect between major, gender, and performance (alpha less than .01). CS = Computer Science, PSY = Psychology, and O = Other.

There were few differences in attitudes that predicted success on fixing problem 2 bugs. There was a significant interaction effect between major and fixing the *generate pvalues* bug for attitudes towards computers: psychology majors who fixed this bug had much lower opinions of computers. There was also a main effect on math attitudes for fixing the *generate valueofoutlier* bug; participants who fixed this bug had higher attitudes towards mathematics.

Programming Strategies

In order to characterize the programming strategies used by participants to solve problem one, each of the variables defined for problem 1 (shown in Table 3 and Table 4) was assessed by the two coders and placed in one or more groups, which defined an aspect of the problem. Table 13 lists these groups, the variables that each group consists of, the Cronbach's alpha of the standardized variables (meaning that each variable was

transformed from its original scale to a standard normal scale), and a description of what the group is attempting to capture. For example, the first group listed, "GUESS AND CHECK", consists of nine of the variables extracted from participants' transcripts, and attempts to capture the degree to which participants' commands are derived through an iterative guess and check process. The reliability of this group, .82, indicates that each of the variables in the group is closely related to the other variables in the group, suggesting that the group is a strongly unitary construct.

Name	Variables in the Group	α	Includes Variables That Reflect the Degree to Which…
Guess and Check	GUESSYN, COMDIVR, COMVOL, RNDLTGT, RNDIN, RND01, GENTO, GENRAND, UNICNT	.82	solutions are derived iteratively
Frustration	SIGHS, CUSSING, TIMEREM, FONTCOM, LAUGHS, USEED	.61	confusion is explicitly expressed through words or behavior
Window Management	WINOFF, WINARR	.81	windows are managed systematically
Use of Specification	READPD, READPDT, UNIFMLY, SRNDNUM, TDTEST, TOTREAD	.86	descriptions and problem boundaries are referenced and used
Desire to Seek Information	READPD, READPDT, WRITE, DRAW, HNOTFND, ERRLINK, HELPTIM, HELPREL, SBUT, COMMENU, SEARCHS, HELPS, HELPDIV, SRCHDIV, LINKUSE, SRNDNUM, HCONTEN, STTST, SPVALUE, CHPWHLP, HAFTTST,	.83	information is pursued
Attention to Feedback	HNOTFND, ERRLINK, OUTREL, MSRLSPD, LISTAFT	.47	feedback from Stata is attended to and used
Syntax Confusion	GUESSYN, COMDIVR, COMVOL, UNMAERR, DUPERR, HLPLIST, USEQSET, USEOBJ, GENOBS, SET1000, GENPAUS, SET4GEN, PARENSP, INPAREN, INVNORM, UNICNT, PAIRED, FRGTCOM, USEED	.65	syntax is improperly constructed or interpreted
Use of GUIs	HELPREL, SBUT, COMMENU, INSPBUT, INSPMEN, BUTUSE, USEED	.72	GUIs are relied on to reach solution
State Comprehension	QHATTMP, FORGETQ, VARWIN, REVWIN, RETBAR, SETWAST, LISTAFT, TTSTUNI, USECLR	.51	the state of artifacts in Stata are understood
Cyclic Behavior	COMDIVR, COMVOL, DUPERR, DUPHELP, DUPSRCH, DUPLINK, UNICNT, USECLR	.80	solutions and methods are repeated unnecessarily
Use of Examples	USETUT, USEEXAM, TUTSET, GENUSE, GENCONS, GENEXAM, USEFUNC, FNINTRO, TTSTEXA, REPTUT, SETSEED	.66	examples from tutorial and online help are references and used
Verbalization	TALKATV, CUSSING, TIMEREM, LAUGH, INQUIS	.61	thoughts about problem and solutions are verbalized

Table 13. Groups of variables used to characterize strategies used by participants to solve problem 1. Cronbach's alphas are reported in the third column, indicating the "relatedness" of the group of variables.

To calculate a quantity to represent each of the groups based on the variables contained within the group, the first principle component from a factor analysis of the variables was extracted, resulting in a standardized variable that placed weights on each component in order to achieve maximum variance. If any of the variables used in the

factor analysis were constant, it was removed from the factor analysis and thus did not contribute to the standardized variable.

What Programming Strategies are Successful?

The first question investigated for problem 1 was how these groups of variables would describe participants that were successful on problem 1 and those who were not. Because each participant began with the same information, but took his or her own path throughout the problem solving session, two methods of exploring this question were used. The first was to compare successful and unsuccessful participants along six evenly divided time intervals of five minutes, to see how the measures changed over time (this data is hereon referred to as "interval data"). Figures showing the standardized measure for each group of variables versus the six time intervals are shown in Figure 20. Each graph represents the average group measure of successful participants (for example, the "Guess and Check" measure) relative to unsuccessful participants. Success in this case was defined as creating the two data sets of uniformly distributed random numbers. Participants who were successful in this sense were not included in intervals following their success.
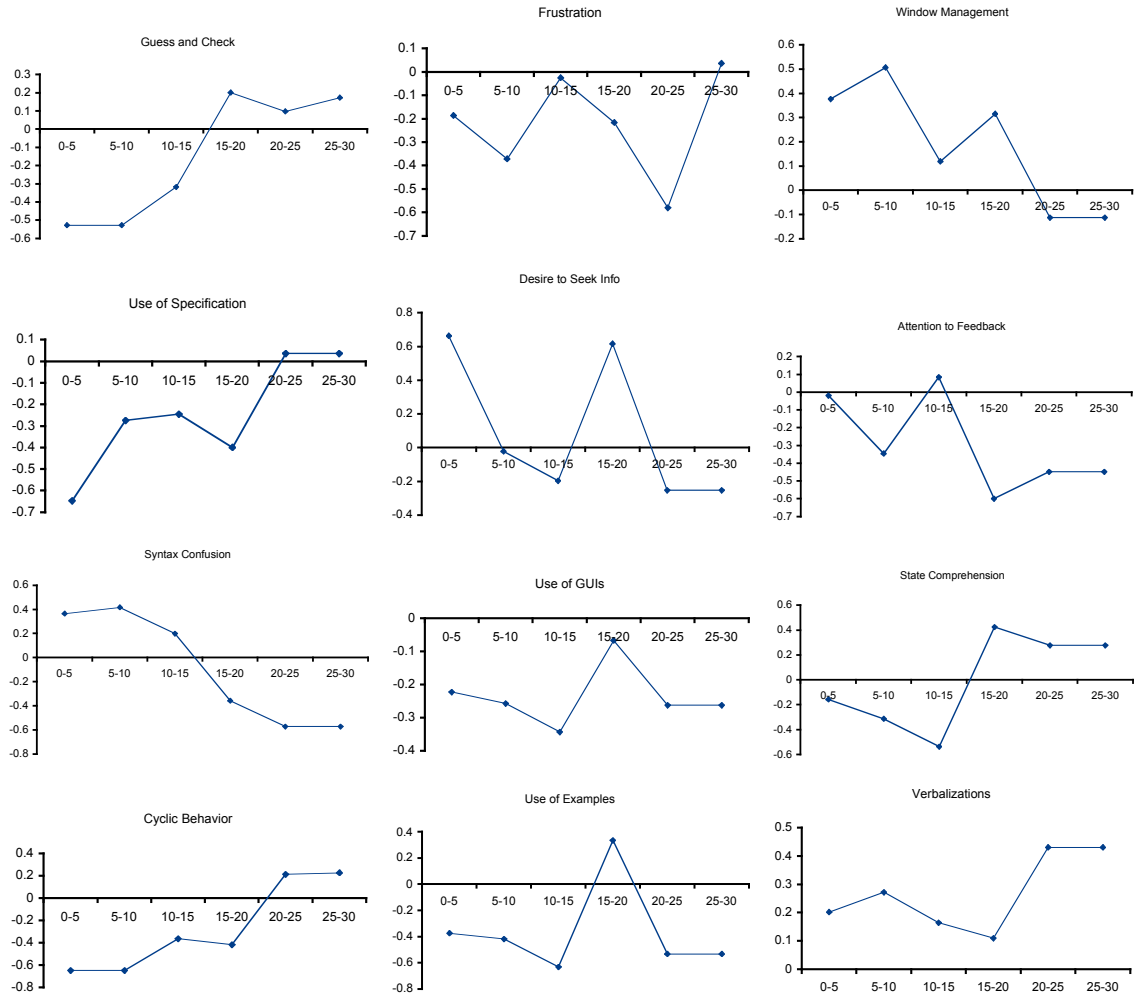
Figure 20. Measures of successful participants relative to unsuccessful participants by intervals of 5 minutes. Positive points mean that successful participants exhibited more of the measure while negative means successful participants exhibited less.

By interpreting the graphs, we can see many trends. Successful participants:

- Exhibited more guess and check behavior as time ran out;

- Were on the whole, less frustrated throughout the problem solving session;

- Performed more window management in the beginning;

- Read the problem specification much less early on;

- Sought more information early on;

- Attended less to errors, likely because fewer errors were made;

- Struggled with syntax early on but overcame it later;

- Used fewer graphical user interfaces throughout the problem;

- Exhibited less cyclic problem solving behavior, but more in time;

- Referenced fewer examples; and

- On the whole, verbalized their thoughts.

The second method used to describe the strategies of successful participants looked at each participants interaction from the beginning until they created the two sets of uniformly distributed random numbers or ran out of time (this data is hereon referred to as "milestone data"). This data, shown in Figure 21 as descriptions of failing participants relative to successful participants, paints a slightly different picture.
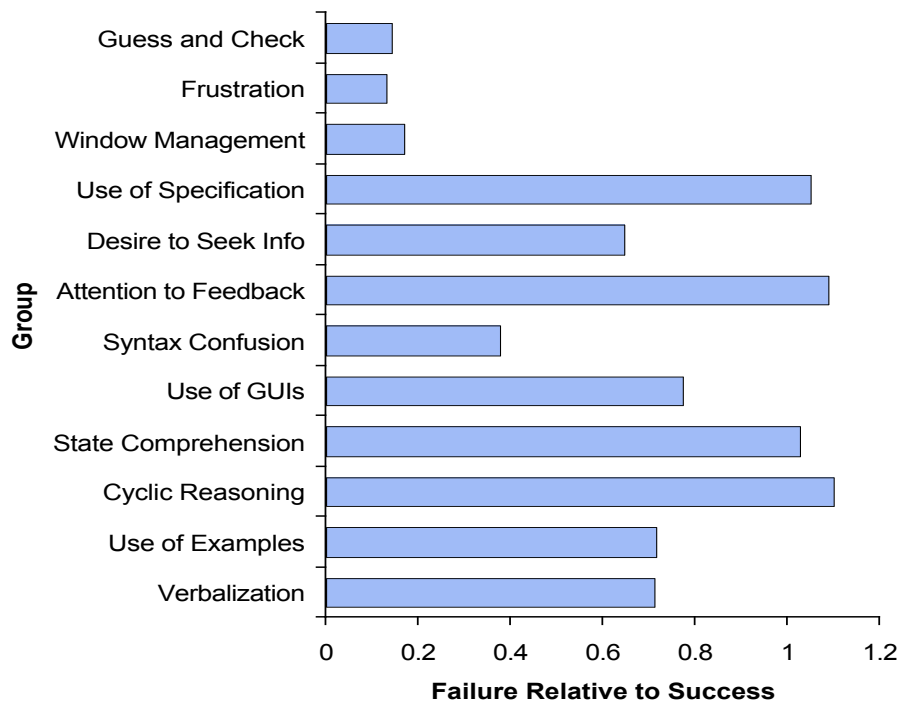
Figure 21. Mean group measures of failing participants relative to successful ones. Values for each group represent the degree to which participants who failed exhibited more of each group behavior.

From this graph, we see that participants who failed to create the two sets of random numbers

- Did slightly more guessing and checking;

- Were slightly more frustrated;

- Did slightly more window management;

- Referenced the specification much more;

- Sought more information;

- Attended more to feedback regarding errors and such;

- Exhibited more syntax confusion;

- Used graphical user interfaces more;

- Demonstrated comprehension of the environment's state;

- Showed much more cyclic behavior;

- Referenced more examples; and

- Verbalized their thoughts more.

Distinct Programming Strategies

To detect distinct programming strategies regardless of success, a hierarchical cluster analysis was performed on the grouped variables listed in Table 13. As seen in the earlier analysis of these variables with respect to success, interval data as well as data up until the first milestone seemed to generate interest results, albeit different ones. Thus both of these sets of data were used in the cluster analysis. However, only the first five minutes of the interval data was used, since it was likely that in the following 25 minutes, participants were diverge greatly on their strategies and subtasks.

As cluster analysis is largely a non-statistical exploratory tool, care was taken to choose the appropriate distance measure and linkage rule for the type of data being used. Euclidian distance (as opposed to squared Euclidian distance) was chosen as the distance measure in order to lessen the impact of outliers on the clustering. Of each linkage rule tested, Ward's method (which attempts to minimize the sum of squares of any two (hypothetical) clusters that can be formed at each step) generated the cleanest hierarchical clustering. As the technique of hierarchical cluster analysis does not provide any guidelines for interpreting the number of clusters from the results, results of 2, 3, 4, 5, 6, and 7 clusters were generated and compared. The membership counts for each cluster for all six results and both the interval and milestone data are provided in Table 14.

Solution Size	Cluster #	Milestone Data		First Interval Data	
		Size	Percent	Size	Percent
2	1	41	54.7	15	20.0
	2	34	45.3	60	80.0
3	1	41	54.7	13	17.3
	2	23	30.7	60	80.0
	3	11	14.7	2	2.7
4	1	41	54.7	13	17.3
	2	21	28.0	13	17.3
	3	11	14.7	47	62.7
	4	2	2.7	2	2.7
5	1	41	54.7	13	17.3
	2	12	16.0	13	17.3
	3	9	12.0	42	56.0
	4	11	14.7	5	6.7
	5	2	2.7	2	2.7
6	1	41	54.7	13	17.3
	2	12	16.0	12	16.0
	3	9	12.0	42	56.0
	4	6	8.0	1	1.3
	5	5	6.7	5	6.7
	6	2	2.7	2	2.7
7	1	41	54.7	13	17.3
	2	9	12.0	12	16.0
	3	9	12.0	36	48.0
	4	6	8.0	6	8.0
	5	5	6.7	1	1.3
	6	2	2.7	5	6.7
	7	3	4.0	2	2.7

Table 14. Sizes of clusters and percent of participants in each cluster for cluster analyses return 2 to 7 clusters.

In choosing the appropriate number of clusters, it is important to consider the size of each cluster in each of the solutions in Table 14. For example, all of the solutions using the first interval data, except for the solution with two clusters, has very small clusters of size less than five. Such solutions, though they may be meaningful, are hardly helpful in distinguishing participants. In order to analyze the solutions further, correlations between solutions were calculated, shown in Table 15.

	Milestone Data						First Interval Data				
Solution Size	3	4	5	6	7	Solution Size	3	4	5	6	7
2	.902	.870	.852	.805	.792	2	.694	.686	.569	.463	.391
3		.921	.918	.891	.799	3		.884	.861	.802	.735
4			.975	.970	.849	4			.956	.852	.777
5				.987	.819	5				.957	.905
6					.822	6					.977

Table 15. Correlations between solution sizes 2 through 7 from cluster analyses on milestone data for problem 1. All correlations were significant at the .01 level.

All of the milestone data solutions correlate well with each other whereas some of the first interval solutions have weaker correlations. This is likely because of the many solutions with small cluster sizes. Finally, correlations between the solutions generated from milestone data and first interval data were calculated to see if the solutions shared anything in. As seen in Table 16, only the solutions with more clusters have significant relationships, suggesting a small level of detail not captured by the solutions with fewer clusters.

		First Interval Data					
	Solution Size	2	3	4	5	6	7
Milestone Data	2	.187	**.316**	**.325**	**.388**	**.436**	**.438**
	3	.137	.199	.163	.214	**.282**	**.299**
	4	.154	.199	.184	.217	**.266**	**.269**
	5	.169	.166	.163	.203	**.258**	**.275**
	6	.132	.121	.122	.162	.213	**.228**
	7	.070	**.232**	.220	**.266**	**.307**	**.305**

Table 16. Correlations between the interval and milestone data, by each solution. Bold correlations are significant at the .05 level.

In attempt to choose a simple, constructive categorization, the three-cluster solution generated from milestone data was chosen as the best candidate. None of the cluster sizes were unreasonably small, and the information gained by increasing the number of clusters was arguably small (see Table 15). Descriptions of these clusters are shown in Figure 22.
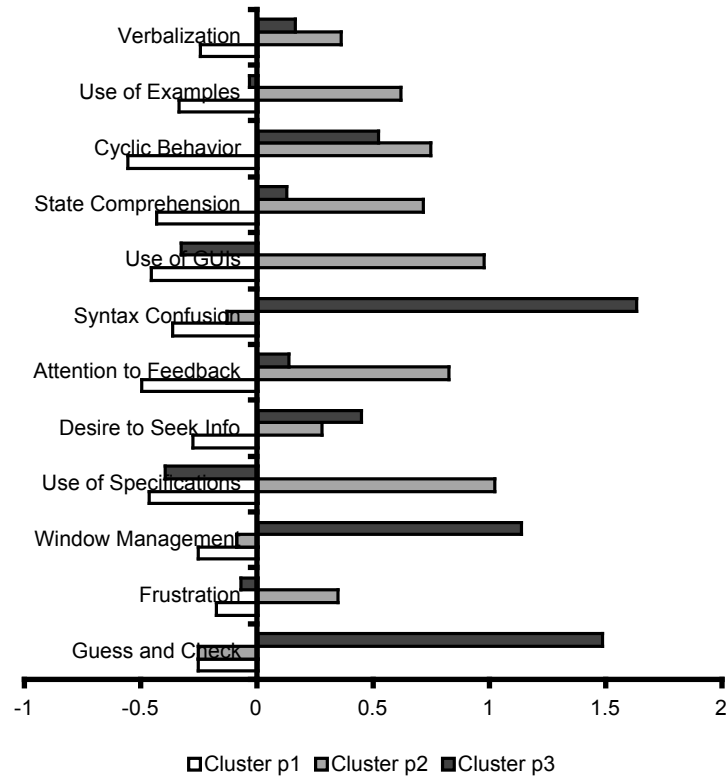
Figure 22. Standardized measures for each cluster, showing relative differences between clusters.

As shown, participants in cluster p1 fell approximately one half standard deviation below the average of all participants for all measures, showing less frustration, guess and check behavior, less use of the problem specification, and so on. Cluster p2 showed above average use of the problem specification, graphical user interfaces, attention to feedback, cyclic behavior, and use of examples. Cluster p3 showed above average measures of syntax confusion, guess and check behavior, window management, and cyclic behavior.

Predictive Power of Programming Strategies

The final question is whether or not these clusters have predictive value. Chi-Squared tests were performed on categorical variables to determine if the clusters were related to performance on problem 1, major, or gender. As seen in Table 17, the three clusters predicted performance on generating random numbers and executing the t-test, as well as major. Those in cluster p1 tended to be computer science majors and successful on the problem; those in cluster p2 tended to be non-computer science majors and largely unsuccessful on the problem; finally, cluster p3 tended to be non-computer science majors and somewhat successful on the problem.

Cluster #	Created the Data Set		Generated Random Numbers		Executed the T Test		Major			Gender	
	True	False	True	False	True	False	Psych.	Comp. Sci.	Other	Female	Male
p1	28	3	34	7	28	13	5	21	15	15	26
p2	21	2	1	22	3	20	12	2	9	12	11
p3	11	0	3	8	3	8	3	1	7	7	4

Table 17. Frequencies of performance variables, major, and gender, by the three clusters. Chi-squared tests were used to test for differences among the groups; shaded variable columns were significant at the .01 level.

The clusters were also related to continuous background variables, as shown in Table 18. Cluster p1 tended to be more intelligence, have greater self-reported experience and ability with mathematics and computer programming, as well as more positive attitudes towards mathematics and computer programming. Cluster p2 tended to be of less intelligence and of lower experience and attitudes towards mathematics and computer programming; finally, cluster p3 tended to be of higher intelligence, of average mathematics experience, of much less programming experience and neutral attitudes towards computers and computer programming.

	Cluster Number		
	p1	p2	p3
Background Characteristic	Mean (Standard Deviation)		
Percent Correct			
Vocab27	.58 (.13)	0.42 (.15)	0.56 (.14)
Vcog I	.64 (.12)	0.51 (.09)	0.63 (.09)
Statistics Test	.35 (.15)	0.27 (.17)	0.36 (.17)
Self-reported experience (0=none, 1=least, 7=most)			
Mathematics experience	5.35 (1.25)	3.39 (1.60)	4.59 (1.82)
Statistics Software Experience	1.31 (1.02)	1.73 (1.45)	1.73 (1.07)
Programming Experience	2.80 (1.85)	1.24 (1.95)	0.79 (.97)
Computer Experience	6.05 (.84)	5.50 (1.15)	5.55 (1.04)
Attitudes Towards… (1=negative, 7 = positive)			
Experiment	5.01 (1.08)	4.77 (.88)	4.80 (.77)
Computers	5.82 (1.21)	4.46 (1.22)	4.73 (1.01)
Statistics	3.30 (1.17)	3.16 (1.15)	3.82 (1.43)
Mathematics	5.23 (1.30)	3.61 (1.78)	4.50 (1.41)
Age (years)	24.46 (4.89)	23.04 (5.24)	27.55 (10.6)
Sleep (hours)	6.85 (1.51)	6.78 (1.70)	6.64 (.81)

Table 18. Means and standard deviations of measures of individual differences by the three clusters. Shaded rows are significant at the .01 level as evidenced by one-way ANOVAs.

Finally, frequencies of combinations of success at the three milestones in problem 1 for each cluster are given in Table 19. Cluster p1 tended to be composed of participants who succeed at the whole problem. Cluster p2 tended to have individuals who were successful only at creating the data set of 1000 objects. Cluster p3 tended to be similar to cluster p2, but had a higher proportion of participants successful at the whole problem.

			Cluster		
Created the Data Set of 1000 Objects	**Created Two Uniformly Distributed Sets of Random Numbers**	**Executed the T Test**	p1	p2	p3
Failure	Failure	Failure	1	2	0
		Success	0	0	0
	Success	Failure	1	0	0
		Success	1	0	0
Success	Failure	Failure	5	17	7
		Success	6	1	1
	Success	Failure	1	3	1
		Success	26	0	2

Table 19. Frequencies of each combination of milestone success for each cluster.

Testing and Debugging Strategies

In order to explore the testing and debugging strategies participants used in problem 2, the same process used for problem 1 was used to generate groups of variables that attempted to measure various aspects of participants' problem solving. Table 20 lists these groups, the variables in the group, reliabilities of the standardized scales, and descriptions of the groups. All of the groups used for problem 1 were used in problem 2, since participants could use everything in the Stata environment they used in problem 1 to solve the problem; however, the variables and descriptions are repeated here as addition variables extracted from problem 2 were included in these groups.

Name	Variables in the Group	α	Includes Variables That Reflect the Degree to Which…
For Command	FORTIME, LPLINE, REPLNUM, REPLX, ROFPHLP, X2Y, CHNFOR5, HLPFOR, HLPSLSH	.63	for command was modified to comply to specification
Changed Words	MODCOMW, REPLNUM, REPLX, X2Y, REMUNDR, GRPTIT, GRPCOM, CHWORD	.60	non-numerics were changed to comply to specification
Frustration	SIGHS, CUSSING, TIMEREM, FONTCOM, LAUGHS, IMPAT	.30	confusion is explicitly expressed through words or behavior
Graph	GRATIME, T2VGRP, SIZEGRA, TRACEGR	.76	graph was used in testing and debugging the do-file
Window Management	WINOFF, WINARR	.96	windows are managed systematically
Changed Comments	EDITCOM, MODCOMW, MODCOMN, GRPCOM, FIXSEMI	.43	comments were changed to comply to specification
Use of Specification	READPD, READPDT, COMTIME, TOPCOM	.55	descriptions and problem boundaries are referenced and used
Attention to Feedback	HNOTFND, ERRLINK, OUTREL, MSRLSPD, DOUSE, RUNUSE, INGERR, ERRS, GRATIME, T2VGRP, SIZEGRA, TRACEGR, NCHB4R	.42	feedback is attended to and used
Familiar Commands	GRNDTIM, CONFGRN, SETTIME, CONFSET, HLPGEN, HLPTTEST	.14	familiar commands were investigated or operated on
Syntax Confusion	GUESSYN, COMDIVR, COMVOL, UNMAERR, DUPERR, HLPLIST, EDITCOM, MODCOMW, MODCOMN, LPLINE, REPLNUM, REPLX, X2Y, CHNFOR5, REMUNDR, GRPTIT, FIXSEMI, HLPGR, HLPFOR, HLPGEN, HLPTTEST, HLPDEL, HLPSLSH	.63	syntax is improperly constructed or interpreted
Use of Commands	TYPOS, COMDIVR, COMVOL, EXMAN, LOOKDAT, TRYCLR	.90	commands were used in testing and debugging the do-file
State Comprehension	QHATTMP, FORGETQ, VARWIN, REVWIN, RETBAR, LOOKDAT, TRYCLR	.97	the state of artifacts in Stata are understood
Use of Examples	USETUT, USEEXAM, CONFGRN, CONFSET	.23	examples from tutorial and online help are references and used
Valueofoutlier Command	GVLTIME, REMUNDR, GVL210, GVL2TH, GVLCH, HLPGEN, HLPSLSH	.59	generate valueofoutlier command was modified to comply to specification
Graph Command	GRPTIME, GRPTIT, GRPCOM, HLPGR	.31	graph command was modified to comply to specification
Guess and Check	GUESSYN, COMDIVR, COMVOL, COMOUT, GNP210, GNP2TH, GNPCH, GVL210, GVL2TH, GVLCH, DOUSE, RUNUSE, REPCHNG	.67	solutions are derived iteratively

Pvalues Command	GNPTIME, GNP210, GNP2TH, GNPCH, HLPGEN, HLPSLSH	.46	generate pvalues command was modified to comply to specification
Desire to Seek Information	READPD, READPDT, WRITE, DRAW, HNOTFND, ERRLINK, HELPTIM, HELPREL, SBUT, COMMENU, SEARCHS, HELPS, HELPDIV, SRCHDIV, LINKUSE, ROFPHLP, TRACEGR, HLPGR, HLPFOR, HLPGEN, HLPTTEST, HLPDEL, HLPSLSH, HLPDO	.71	information is pursued
Use of GUIs	HELPREL, SBUT, COMMENU, INSPBUT, INSPMEN, BUTUSE2	.44	GUIs are relied on to reach solution
Cyclic Behavior	COMDIVR, COMVOL, DUPERR, DUPHELP, DUPSRCH, DUPLINK, REPCHNG, TRYCLR	.43	solutions and methods are repeated unnecessarily
Change Range	MODCOMN, CHNFOR5, GNP210, GNP2TH, GNPCH, GVL210, GVL2TH, GVLCH, CHNUM	.75	ranges were changed to comply to specification
Verbalization	TALKATV, CUSSING, TIMEREM, LAUGHS, INQUIS	.55	thoughts about problem and solutions are verbalized

Table 20. Groups of variables used to characterize strategies used by participants to solve problem 2. Cronbach's alphas are reported in the third column, indicating the "relatedness" of the group of variables.

As with problem 1, quantities representing each of the groups based on the variables contained within the group were created by extracting the first principle component from a factor analysis of the variables

What Testing and Debugging Strategies Are Successful?

As with problem 1, these variables were used to fist describe participants that were successful on problem 2 and those who were not. Interval data and milestone data was used for problem 2 as well, with the first milestone defined as when the participant stopped working on problem 2 (when the participant believed the testing and debugging was complete). Figures showing the standardized measure for each group of variables for problem 2 versus four time intervals of five minutes are shown in Figure 23 through 27. As before, each graph represents the average group measure of successful participants (for example, the "Guess and Check" measure) relative to unsuccessful participants. Success in this case was plotted separately for each of the four bugs.

Frustration   **Verbalizations**

Generate *valueofoutlier* Range   Generate *pvalue* Range   Fixed Inverted Axes   Fixed Loop Iteration
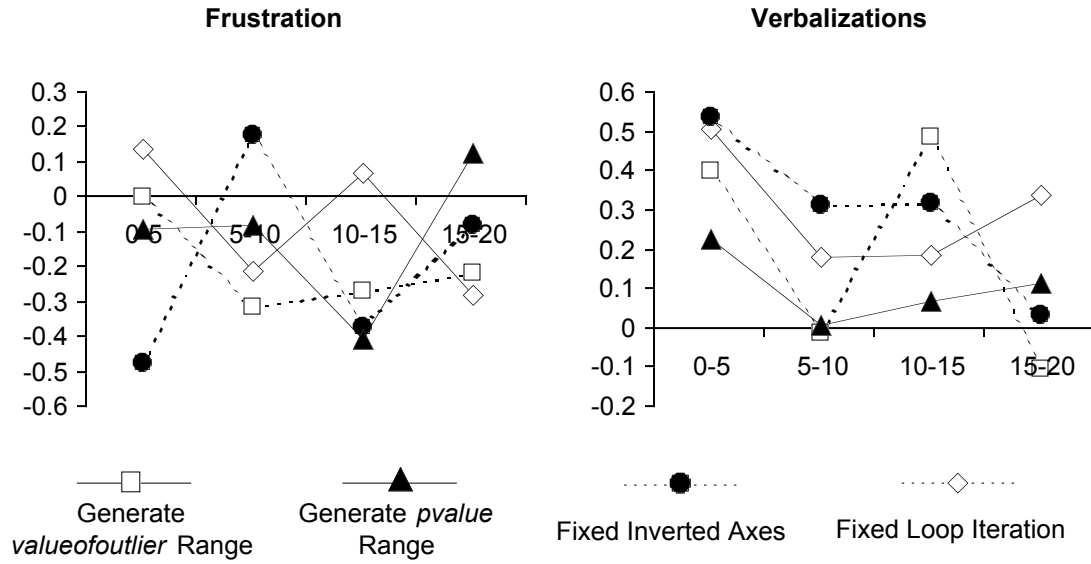
Figure 23. External environment measures for success relative to failure, for each bug in problem 2.

As seen in Figure 23, participants differed greatly for each bug. Successful participants for all bugs were on average less frustrated throughout problem 2. Furthermore, participants who were successful on each of the bugs tended to verbalize their thoughts more than unsuccessful participants.

Graph Inspection

Attention to Feedback

Use of GUIs

Window Management

State Comprehension

Use of Specifications

Generate *valueofoutlier* Range

Generate *pvalue* Range
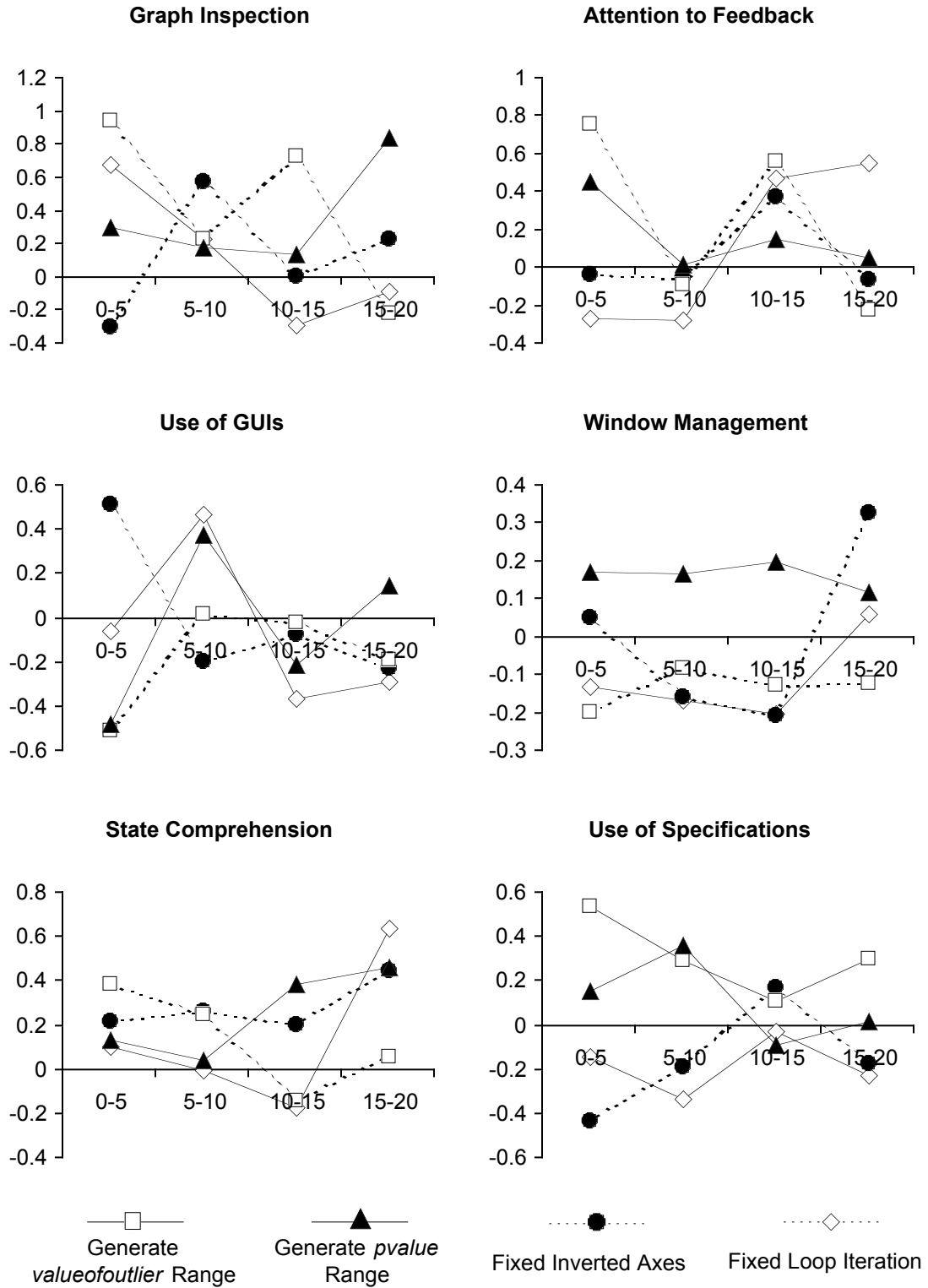
Fixed Inverted Axes

Fixed Loop Iteration

Figure 24. Measures for success relative to failure, for each bug in problem 2.

As seen in Figure 24, successful participants on each bug inspected the graph more than unsuccessful participants, and also attended to feedback provided by Stata more. Participants did not differ too much on how much they used graphical user interfaces, however, those successful at fixing the *generate pvalue* bug did more window management, and those successful on the other three bugs did less window management. On average, successful participants on each bug comprehended the state of Stata more than unsuccessful participants. Those who were successful at fixing the *generate valueofoutlier* range used the problem specification more, while those who were successful on the other bugs used it less.

Desire To Seek Info

Cyclic Behavior

Change Ranges

Changed Comments

Changed Words

Guess and Check

Generate *valueofoutlier* Range

Generate *pvalue* Range
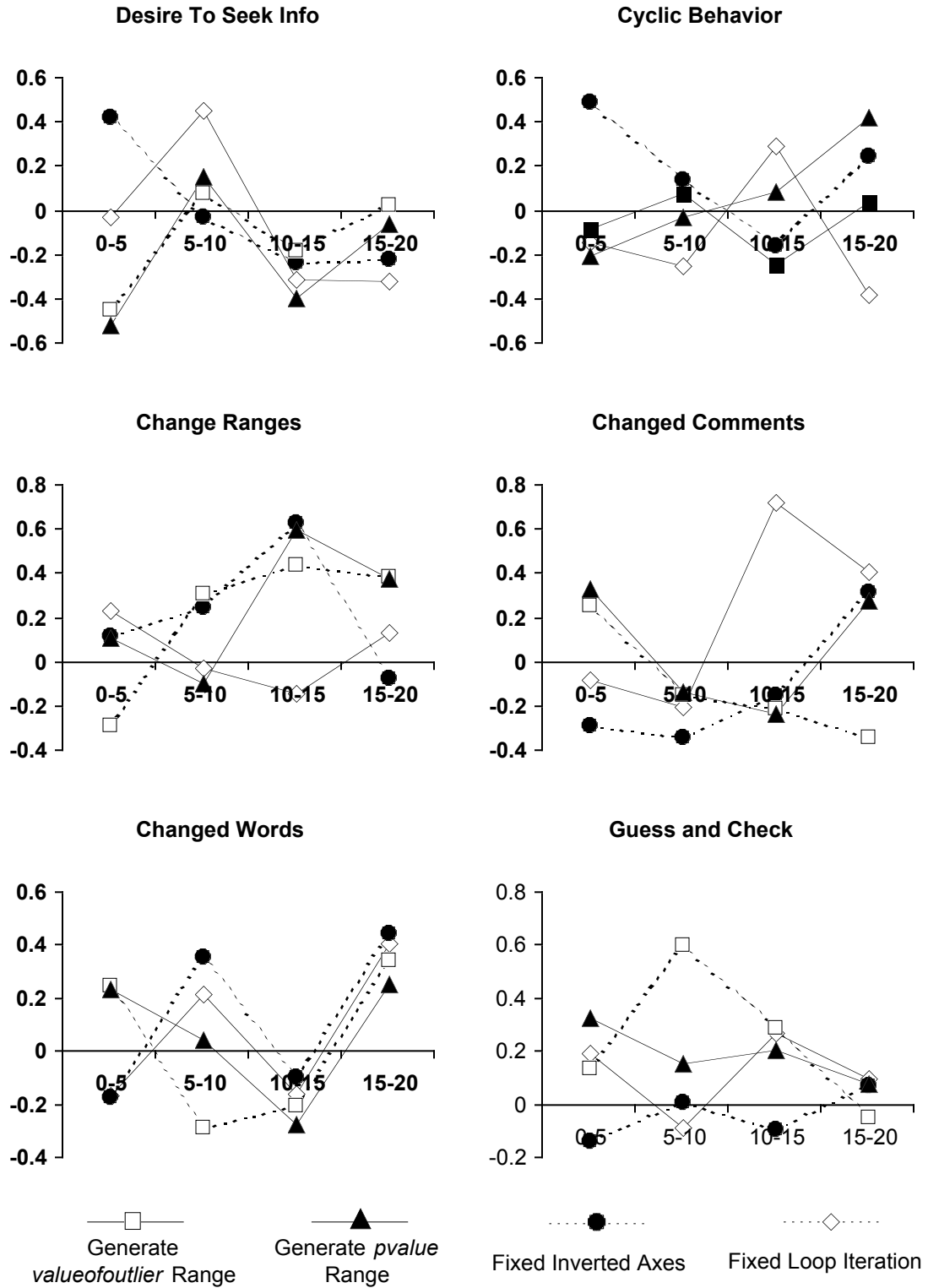
Fixed Inverted Axes

Fixed Loop Iteration

Figure 25. Measures of problem solving behaviors for success relative to failure, for each bug in problem 2.

As seen in Figure 25, participants differed greatly for each bug with respect to problem solving behaviors. Participants successful on fixing the inverted axes spent more time seeking information in the beginning and less at the end; for the other three bugs, successful participants sought information earlier than unsuccessful participants. There were few differences in cyclic behavior except for the inverted axes bug, where successful participants seemed to exhibit more of this behavior early on. As two of the bugs were observation range problems, it is no surprise that participants successful on these two bugs changed more ranges throughout the problem; in fact, participants successful on fixing the inverted axes also changed more ranges. Participants successful at fixing the inverted axes and the loop iteration range were more likely to change comments of the code in the do-file, but those who fixed the *pvalue* and *valueofoutlier* observation ranges were less likely to change comments. During the second and fourth intervals, successful participants on all bugs but the *generate valueofoutlier* bug were more likely to change words in commands, as opposed to numbers. Finally, successful participants exhibited more guessing and checking behavior for the *generate pvalues* bug, more for the *generate valueofoutlier* bug, but no more than unsuccessful participants on the *intervted axes* and *loop range* bugs. Furthermore, successful participants exhibited less guessing and checking behavior on the *generate pvalues* and *generate valueofoutlier* bugs later in the problem.

Use of Commands   **Syntax Confusion**

Generate *valueofoutlier* Range   Generate *pvalue* Range   Fixed Inverted Axes   Fixed Loop Iteration
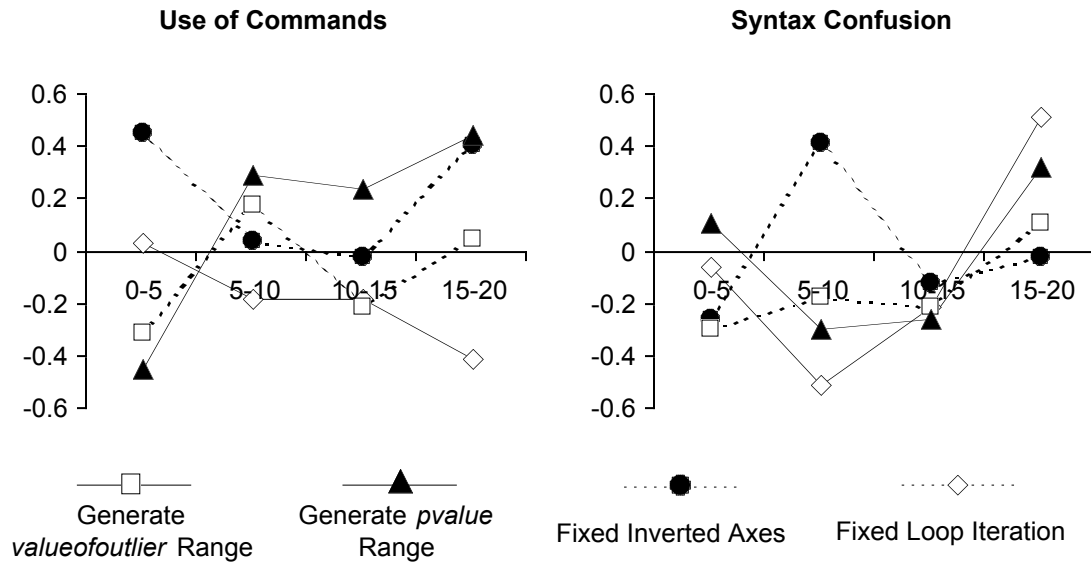
Figure 26. Measures regarding command creation for success relative to
failure, for each bug in problem 2.

As seen in Figure 26, participants successful at fixing the *generate pvalue* bug used
more commands through time, yet participants successful at fixing the other three bugs
used an average amount or fewer commands than unsuccessful participants. Though,
participants successful on all four bugs tended to have less syntax confusion in the
beginning and more later on.

Generate Valueofoutliers Command

Work On Familiar Commands

Generate Pvalues Command

Work On For Command

Generate *valueofoutlier* Range

Generate *pvalue* Range
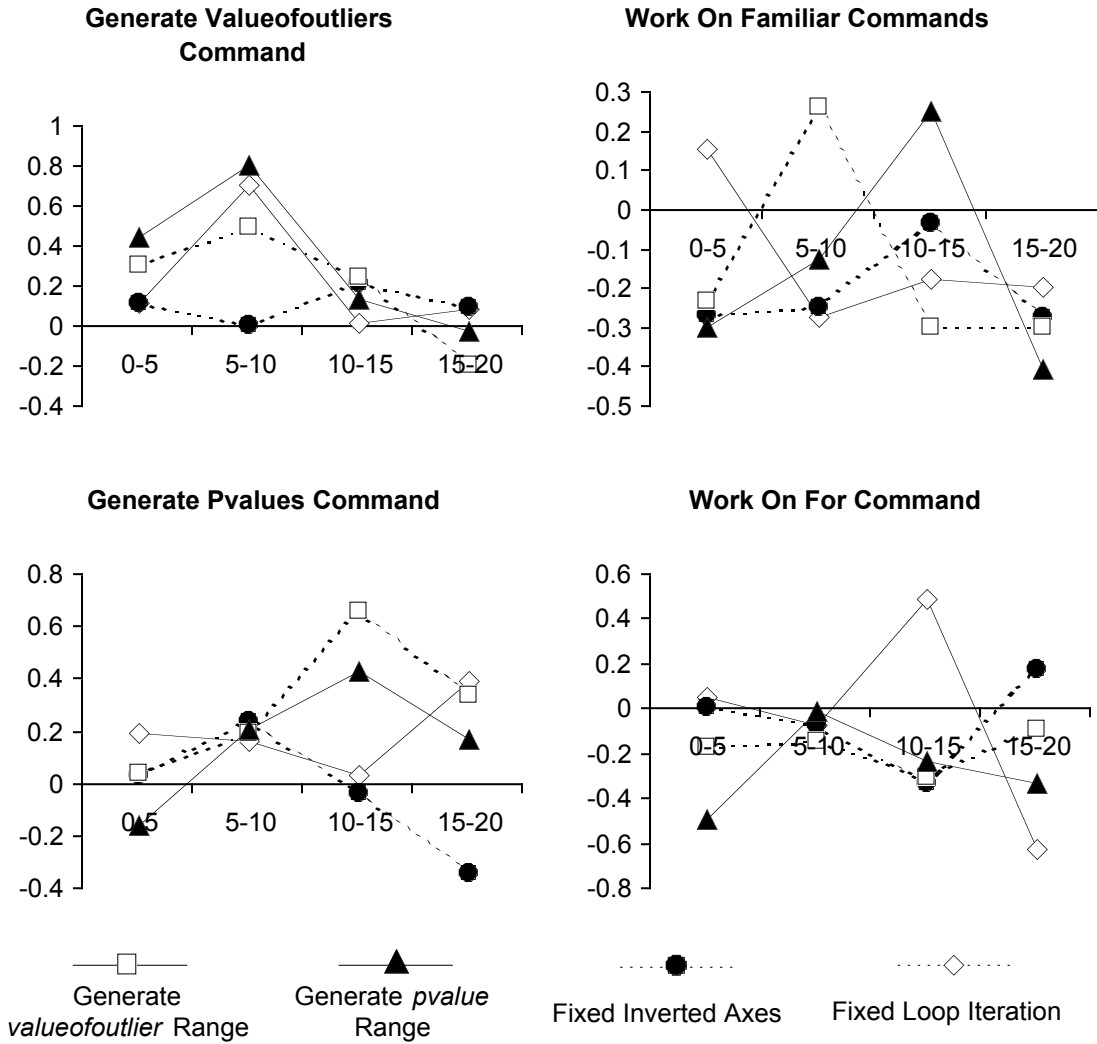
Fixed Inverted Axes

Fixed Loop Iteration

Figure 27. Measures regarding work on the commands in the do-file for success relative to failure, for each bug in problem 2. The graph command was not included because of insufficient data for comparison.

As seen in Figure 27, participants successful at each bug inspected the *valueofoutlier* command more than unsuccessful participants early in the problem. In general, participants successful at each bug inspected familiar commands (such as "set obs 1000" and "generate rand1=uniform()") less than unsuccessful participants. Furthermore, participants who were successful at each bug spent more time inspecting the *generate*

pvalues command. Finally, successful and unsuccessful participants worked on the for

loop command equally.

As with problem 1, data for the first milestone was also inspected using the variables

in Table 20. The milestone for problem 2 was considered to be from the beginning of

participants' interaction until they believed they had found all of the bugs and the do-file

was correct, thus these results included all the data for each participant, unlike the
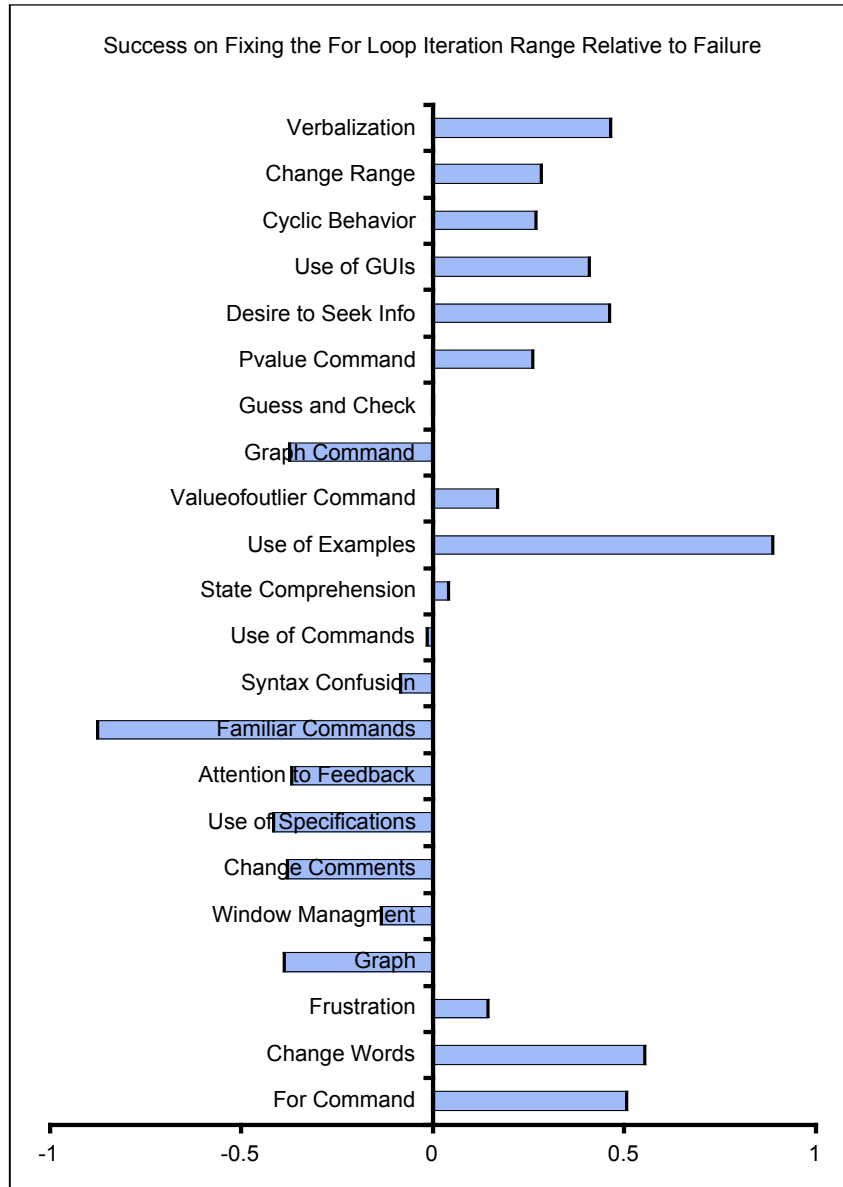
milestone results for problem 1.

Figure 28. Success relative to failure for each of the measures across the whole duration of each participant's interaction.

Participants who successfully fixed the for loop iteration bug, described in Figure 28, exhibited more verbalization of their thoughts, changed more observation ranges, exhibited more cyclic behavior, used more graphical user interfaces, sought more information, spent more time inspecting the *generate pvalues* command, use far more examples, changed more words in commands, and inspected the for loop command more. Furthermore, such participants inspected the graph commands familiar from problem 1

less, as well as attended to less feedback, read the problem specification less, changed

fewer comments, and payed less attention to the graph produced by the do file.

Figure 29. Success relative to failure for each of the measures across
  the whole duration of each participant's interaction.

As seen in Figure 29, participants successful at fixing the inverted axes (of which

there were only 25 percent), exhibited less use of graphical user interfaces, used fewer

examples, and inspected familiar commands less. Furthermore, such participants

inspected the graph command more (which should be expected), but also inspected the

generate valueofoutlier command more, was more confused about syntax, changed more

comments, and inspected the graph produced by Stata more.

Figure 30. Success relative to failure for each of the measures across
the whole duration of each participant's interaction.

Appropriately, participants who successfully fixed the *generate pvalues* command

spent more time inspecting the command and changing observation ranges, as seen in

Figure 30. However, successful participants also inspected the graph more, exhibited

more cyclic behavior, and were more confused with syntax. They also attended to

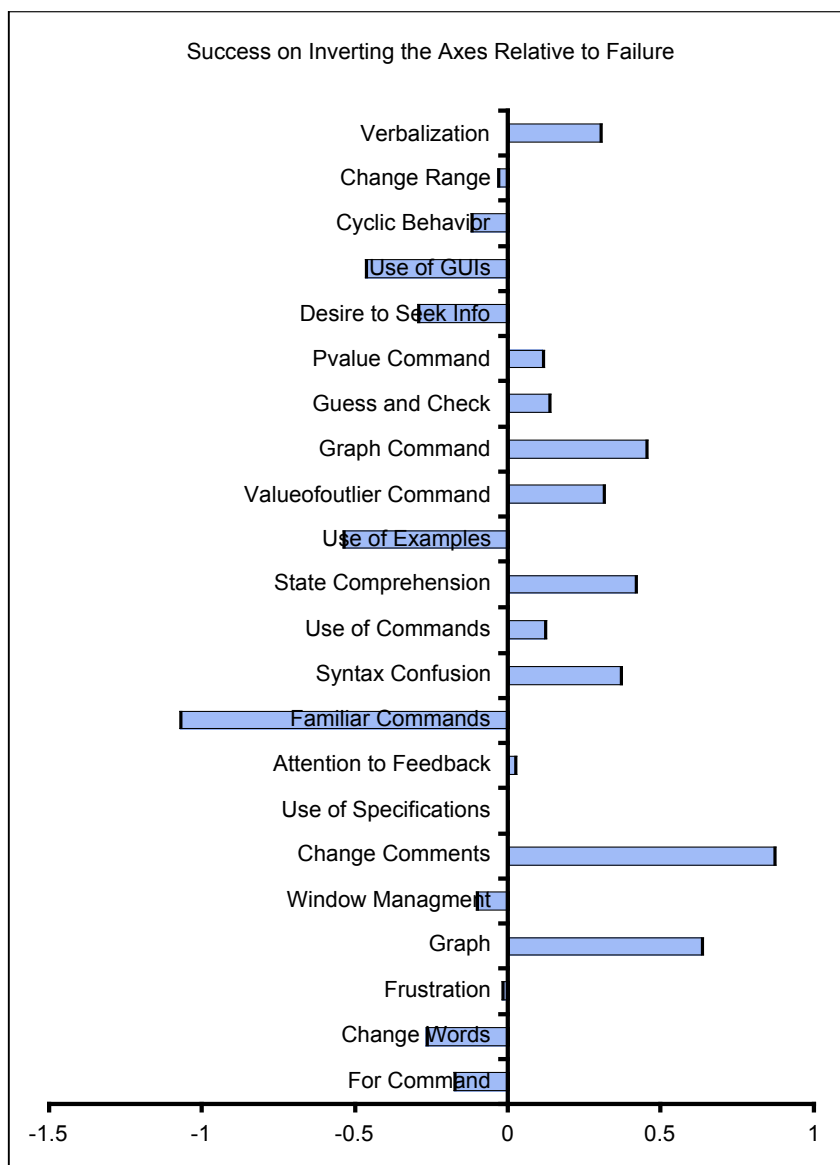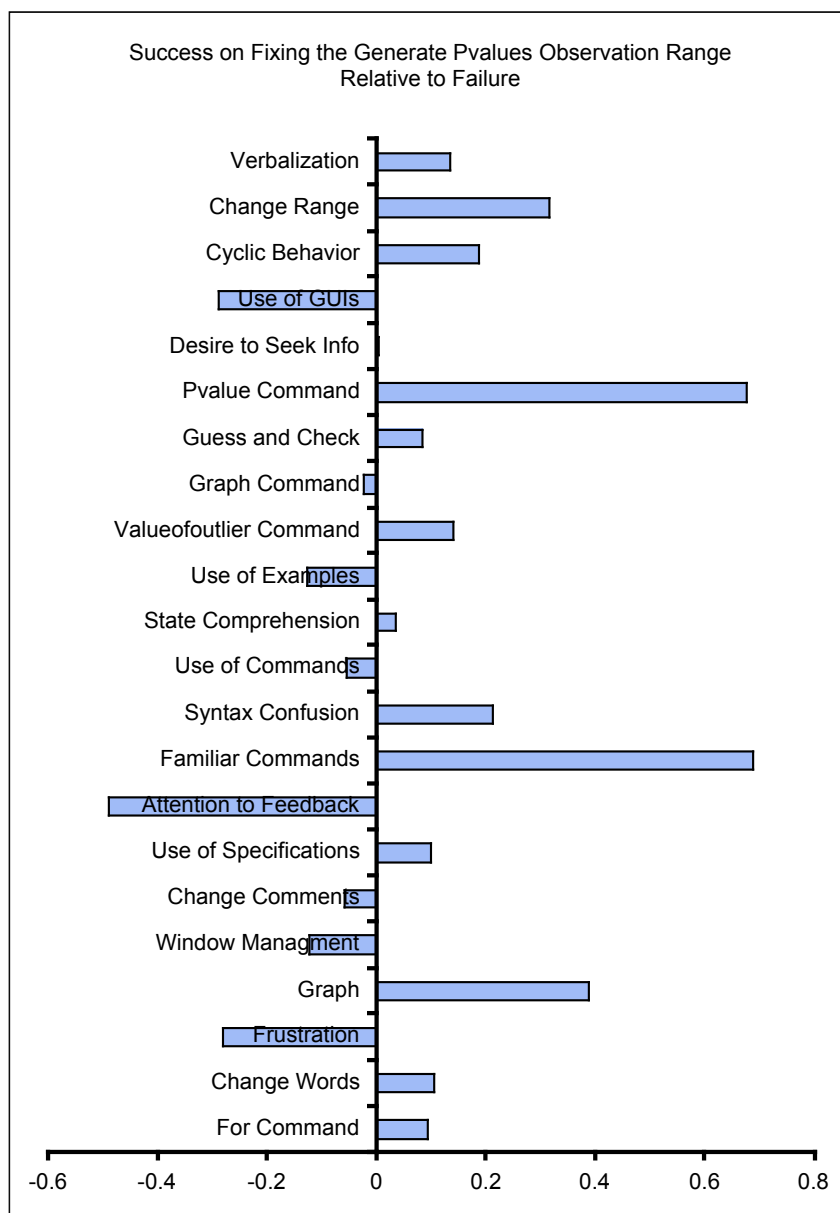feedback from Stata less, were less frustrated, and used graphical user interfaces more.
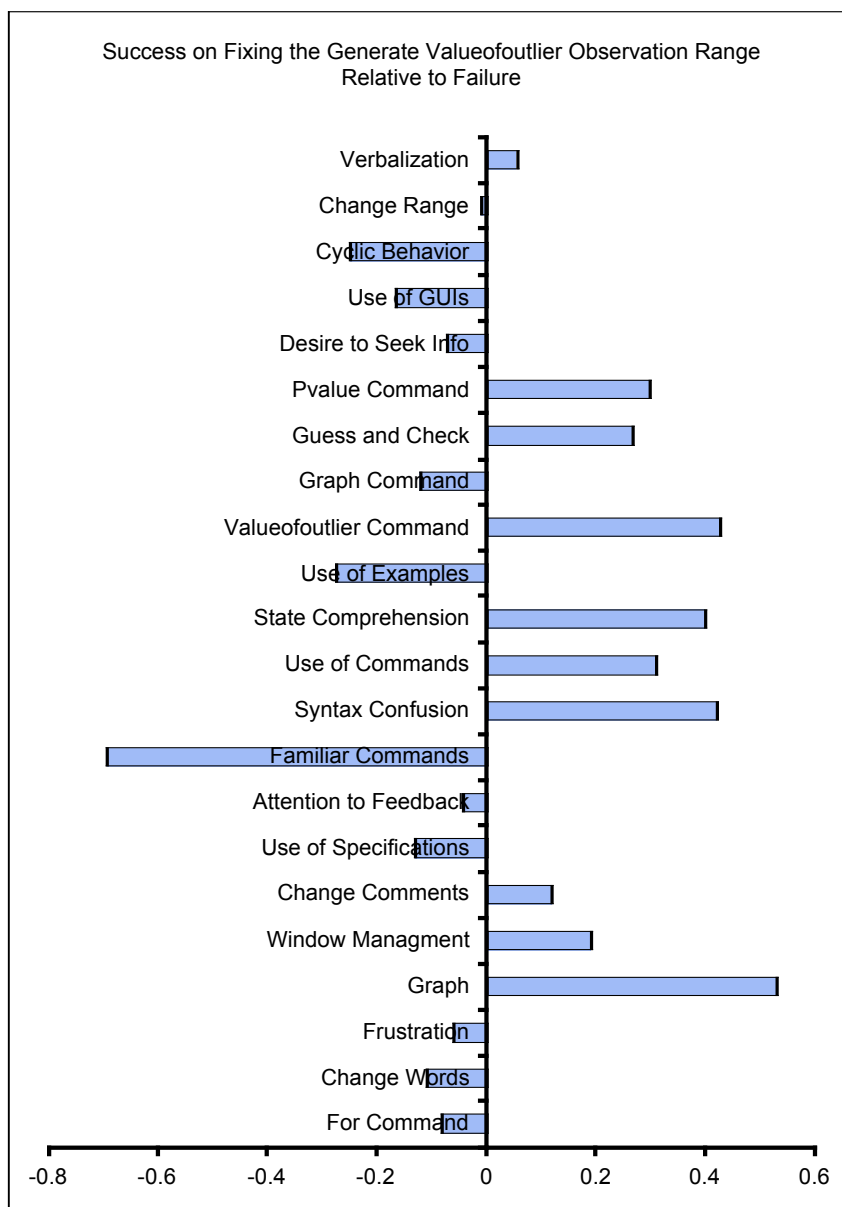

Figure 31. Success relative to failure for each of the measures across
the whole duration of each participant's interaction.

As we see in Figure 31, participants who successfully fixed the observation range in

the *generate valueofoutlier* command appropriately inspected the command itself more.

Such participants also spent more time viewing the graph and managing windows, and

inspecting the *generate pvalues* command. Furthermore, these participants used more commands and were confused by them, performed more guess and check behavior, and comprehended the environment's state more. Lastly, these successful participants inspected familiar commands far less and used fewer examples from the online help.

Distinct Testing and Debugging Strategies

To detect distinct programming strategies irrespective of success, a hierarchical cluster analysis was performed on the grouped variables listed in Table 20, just as with problem 1. Also as with problem 1, earlier analysis of these variables with respect to success, interval data as well as data up until the first milestone seemed to generate different results, thus both of these sets of data were used in the cluster analysis. Just as with problem 1, it was likely that past the first five minutes, participants would diverge greatly on their strategies and subtasks, thus only the first five minutes of problem 2 were used for the interval data. Ward's method combined with Euclidian distance again produced the cleanest clusters; solution sizes of 2 to 7 clusters were generated for milestone data and interval data.

The six solutions for the each data set are provided in Table 21. Again, as with the cluster analyses performed for problem 1 data, when more than 3 clusters were generated, the size of the fourth group was often too small to be practically useful.

		Milestone Data		First Interval Data	
Solution Size	Cluster #	Size	Percent	Size	Percent
2	1.00	60	80.0	45	60.0
	2.00	15	20.0	30	40.0
3	1.00	20	26.7	45	60.0
	2.00	40	53.3	24	32.0
	3.00	15	20.0	6	8.0
4	1.00	20	26.7	45	60.0

		Milestone Data		First Interval Data	
	2.00	40	53.3	21	28.0
	3.00	9	12.0	3	4.0
	4.00	6	8.0	6	8.0
	1.00	20	26.7	45	60.0
	2.00	40	53.3	20	26.7
5	3.00	6	8.0	3	4.0
	4.00	6	8.0	6	8.0
	5.00	3	4.0	1	1.3
	1.00	20	26.7		60.0
	2.00	25	33.3	9	12.0
6	3.00	15	20.0	3	4.0
	4.00	6	8.0	11	14.7
	5.00	6	8.0	6	8.0
	6.00	3	4.0	1	1.3
	1.00	20	26.7	27	36.0
	2.00	22	29.3	18	24.0
	3.00	15	20.0	9	12.0
7	4.00	6	8.0	3	4.0
	5.00	6	8.0	11	14.7
	6.00	3	4.0	6	8.0
	7.00	3	4.0	1	1.3

Table 21. Sizes and percent of participants in each cluster for cluster analyses producing 2 to 7 clusters.

As seen in Table 22, generating a fourth cluster from the milestone data resulting in a higher correlation than from two clusters to three, suggesting that some detail is lost by only choosing three clusters. The first interval data shows similar effects. Unlike problem 1, Table 23 shows that clusters generated from the milestone data did not correlate with the first interval data.

Milestone Data						First Interval Data					
Solution Size	3	4	5	6	7	Solution Size	3	4	5	6	7
2	.784	.825	.846	.836	.645	2	.919	.822	.804	.856	.877
3		.958	.923	.916	.786	3		.964	.918	.908	.907
4			.926	.914	.770	4			.936	.860	.851
5				.963	.802	5				.900	.881
6					.801	6					.973

Table 22. Correlations between different size solutions for the milestone data and interval data. All correlations were significant at the .01 level.

	Solution Size	Milestone Data					
		2	3	4	5	6	7
First Interval Data	2	.136	-.080	.052	-.022	-.036	-.047
	3	.042	-.049	.038	-.028	-.042	-.063
	4	.075	.044	.131	.056	.041	-.002
	5	.048	.044	.120	.048	.023	.107
	6	.127	.109	.183	.097	.079	.120
	7	.094	.081	.164	.062	.051	.089

Table 23. Correlations between interval data and milestone data solutions of the same size. None of the correlations were significant at the .05 level.

As with problem 1, the purpose of the cluster analysis was to detect simple, constructive categories, thus the three-cluster solution generated from milestone data was chosen as the best candidate. None of the cluster sizes were unreasonably small, and at face value, the milestone data would seem to have higher face validity than the first five minutes of participants' interactions since participants could vary more in what they did than in problem 1. Descriptions of these three clusters are shown in Figure 32.

As shown in the figure, cluster d1 tended to be participants that viewed the graph less,

changed fewer comments, inspected more familiar commands, used more examples,
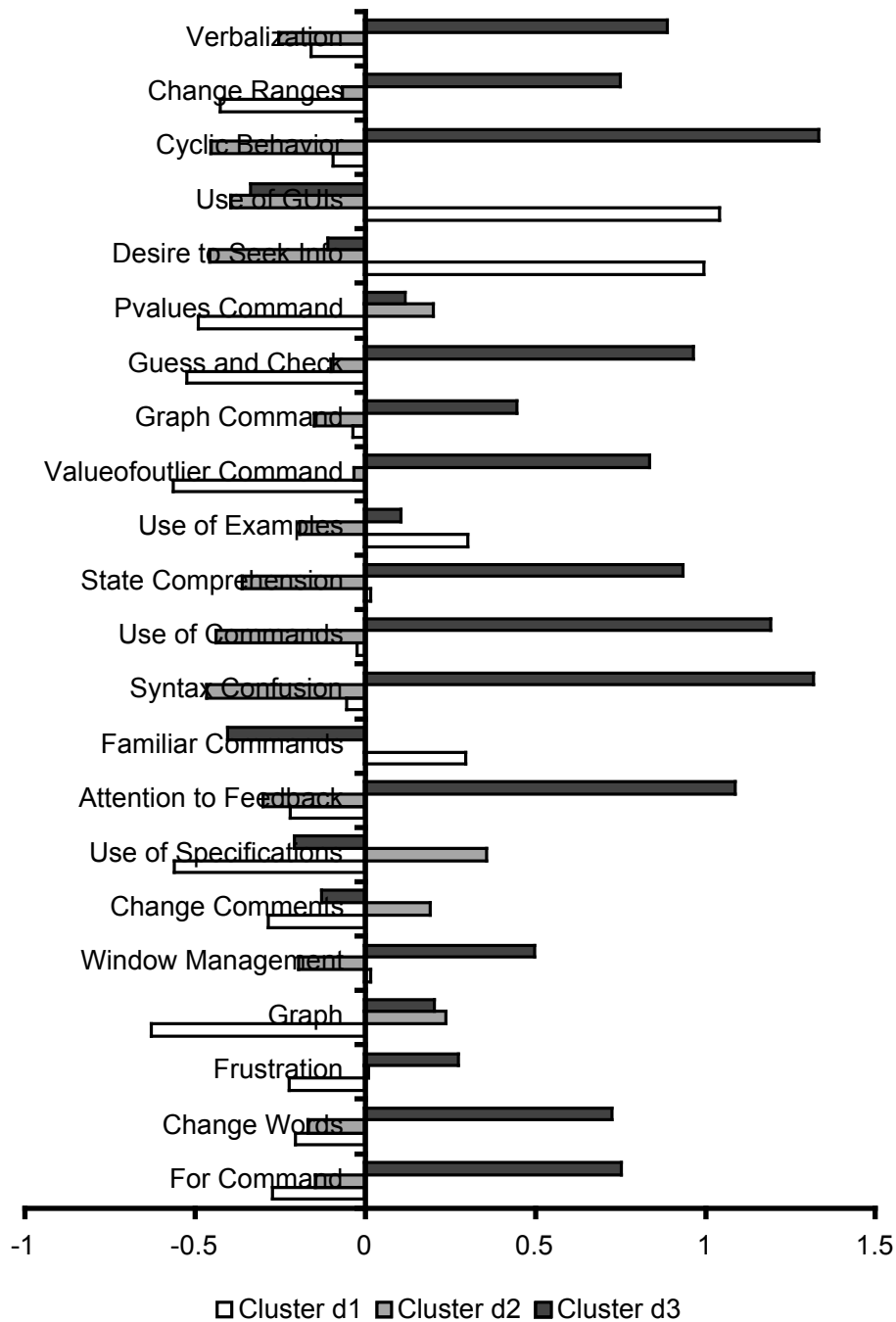

Figure 32. Descriptions of the three clusters generated from the
  problem 2 milestone data.

sought more information, used more graphical user interfaces, while inspecting the pvalues and valueofoutlier commands less than the other two groups. Cluster d1 participants also exhibited less guess and check behavior and changed ranges less than participants in other clusters. Participants in cluster d2 tended to be more average, exhibiting slightly less cycle behavior, slightly less use of graphical user interfaces, slightly less use of commands (and thus less syntax confusion) and slightly less comprehension of the state of the Stata environment. Finally, cluster d3 was the above average group on most measures: they spent more time on the for-loop, changed more words in commands, performed more window management, payed attention to more feedback from Stata. Participants in this group also used more commands and thus experienced more syntax confusion, verbalized their thoughts far more, exhibited more cyclic behavior, and inspected the *valueofoutlier* command and graph commands more. They also used graphical user interfaces slightly less.

Frequencies of combinations of bugs fixed for each cluster are given in Table 24. Cluster d1 tended to be composed of participants who either fixed not bugs, or just the for loop iteration bug. Cluster d2 tended to have individuals who fixed the *generate pvalue* and *generate valueofoutlier* bugs, and sometimes the inverted axes and for loop iteration bug. Cluster d3 tended to have individuals who fixed the *generate pvalue* bug and the inverted axes bug.

				Cluster		
Generate pvalue	*Generate valueof...*	**Inverted Axes**	**For Loop Iteration**	1	2	3
Failure	Failure	Failure	Failure	7	6	1
			Success	4	2	1
		Success	Failure	0	1	0
			Success	0	0	0
	Success	Failure	Failure	2	2	0
			Success	0	0	1
		Success	Failure	0	1	0
			Success	0	0	0
Success	Failure	Failure	Failure	2	5	3
			Success	1	5	1
		Success	Failure	1	2	2
			Success	1	1	1
	Success	Failure	Failure	1	8	1
			Success	1	3	2
		Success	Failure	0	4	1
			Success	0	0	1

Table 24. Frequencies of combinations of bugs fixed for each cluster.

Finally, in order to see if the clusters from problem 1 have anything in common with the clusters for problem 2, tabulations by the two problems are listed in Table 25. Cluster p1 participants from problem 1 tended to be equally distributed among the clusters in problem 2. Cluster p2 participants tended to belong mostly to cluster d2. Cluster p3 participants tended to belong to cluster d3.

		Problem 2		
	Cluster #	d1	d2	d3
Problem 1	p1	16	15	10
	p2	2	16	5
	p3	2	9	

Table 25. The relationship between the clusters from problem 1 and the clusters from problem 2.

Predictive Power of Testing and Debugging Strategies

These three clusters were analyzed for their predictive value, as with problem 1. Chi-Squared tests were performed on categorical variables to determine if the clusters were

related to performance on problem 2, major, or gender. As seen in Table 26, the three

clusters were only related to major and successful fixing of the *generate valueofoutlier*

command. Those in cluster d1 tended to be non-psychology majors and successful at

fixing the *valueofoutlier* bug (and though not significant, were not successful on the other

three bugs); those in cluster d2 tended to be non-computer science majors and largely

unsuccessful on each of the bugs; finally, cluster d3 tended to be non-psychology majors

and more unsuccessful on each bug than successful.

Cluster	Fixed Loop Range		Fixed Inverted Axes		Fixed Pvalue Range		Fixed Valueofoutlier Range		Major			Gender	
	True	False	True	False	True	False	True	False	Psych.	Comp Sci.	Other	f	m
d1	7	13	2	18	4	16	13	7	3	8	9	10	10
d2	11	29	9	31	18	22	12	28	16	8	16	19	21
d3	1	8	5	10	6	9	3	12	1	8	6	5	10

Table 26. Frequencies of performance variables, major, and gender, by the three clusters. Chi-squared tests were used to test for differences among the groups; shaded variable columns were significant at the .01 level.

The clusters were also related to continuous background variables, as shown in Table

27. Only two measures showed significant relationships: programming experience and

attitudes toward the experiment, the latter of which is questionable. Cluster d1 tended to

have medium programming experience, cluster d2 the lowest, and cluster d3 the most;

however, since all three clusters contained computer science majors, it is difficult to

ascertain whether or not programming experience was different for the three clusters. If

in fact the three clusters are meaningful, it may very well be a more complex construct

underlying programming experience.

Background Characteristic	Cluster Number		
	d1	d2	d3
	Mean (Standard Deviation)		
Percent Correct			
Vocab27	.58 (.14)	.50 (.14)	.53 (.18)
Vcog I	.63 (.12)	.58 (.10)	.61 (.15)
Statistics Test	.36 (.14)	.31 (.17)	.34 (.15)
Self-reported experience (0=none, 1=least, 7=most)			
Mathematics Experience	4.93 (1.19)	4.26 (1.83)	5.27 (1.64)
Statistics Software Experience	1.35 (1.05	1.5 (1.29	1.7 (1.07)
Programming Experience	2.52 (1.76)	1.20 (1.62)	3.59 (2.00)
Computer Experience	6.13 (.90)	5.65 (1.10)	5.8 (.75)
Attitudes Towards… (1=negative, 7 = positive)			
Experiment	4.89 (.84)	4.68 (.98)	5.53 (.90)
Computers	5.65 (1.05)	4.76 (1.39)	5.98 (1.05)
Statistics	3.34 (1.07)	3.39 (1.33)	3.18 (1.09)
Mathematics	4.83 (1.27)	4.40 (1.68)	4.95 (1.91)
Age (years)	25.55 (5.92)	24.28 (6.86)	23.6 (4.6)
Sleep (hours)	6.8 (1.64)	6.63 (1.21)	7.27 (1.87)

Table 27. Means and standard deviations for measures of individual differences by the three clusters. Shaded rows are significant at the .01 level, as evidenced by one-way ANOVAs.

Discussion

A wealth of data was generated in this study for which many questions can be investigated. This study focused on two major questions:

- What differences are there among successful and unsuccessful participants with regard to individual differences such as experience, intelligence, attitudes, and problem solving style?

- Do individuals' problem solving styles naturally cluster into distinct categories, and if so, do these categories have any predictive value with regard to individual differences and programming success?

Individual Differences and Performance: Who is Successful?

On the whole, the sample of college students used in the study did not reach a ceiling or a floor on either problem: even the most poorly performing groups still had successful individuals while the most successful group still had unsuccessful individuals. The interesting differences to consider are with respect to each individual milestone in the programming task and the testing and debugging task.

The Programming Task

For example, the very first task for problem one was to type the command "set obs 1000" to generate a data set with 1000 objects; all participants had experience typing the command "set obs 10" during the tutorial, so it is no surprise that nearly participants

successfully completed this task. What is surprising is that five participants did *not* complete this task. Even comparing the means of the seventy participants who did to the only five who did not, we do not see large differences in intelligence, attitudes, or experience. Quite possibly, these were individuals who failed on problem 1 completely and saved the first step for last.

The second task in problem 1 was arguably the most important and the most telling: to create two variables with uniformly distributed random numbers, participants needed to be able to search for information in Stata, to have the patience to read numerous help documents, to know which documents of hundreds to read, and to know when they found what they were looking for. Once they did, they needed to be able to construct the command "generate *somename* = uniform()" by interpreting the description of the uniform() function in the help file. Furthermore, the picky syntax in Stata would produce ambiguous syntax errors to the effect of "invalid syntax" if a participant put anything—numbers, spaces, words—in the parentheses in the command, thus they needed to have some sort of intuition about what could be wrong with their attempts. Given the complicated nature of the task, who were the individuals that succeeded? They tended to be more intelligent and have much higher mathematics and programming experience, suggesting that there some problem solving skills intrinsic in these categories of knowledge that allowed the successful participants to eventually construct the required command.

Because this task was the main barrier to success for most participants, it is also interesting to consider the aspects of their problem solving captured by the groups of variables presented earlier in Table 13, relative to success of creating the random

numbers. Not only were successful participants more intelligent, but they also exhibited some arguably better strategies. If the problem is framed as a search problem in which participants were presented with a large search space and required to find a single solution buried deep within the online help, the most appropriate strategy would be to search and search with purpose. Successful participants performed less guess and check behavior, sought more information early on, and repeated searches and commands far less than unsuccessful participants. All of these aspects of their problem solving demonstrate why unsuccessful participants were more frustrated: they repeated searches, guessed syntax and read the problem specification. In the context of this search problem, these unsuccessful participants were effectively stuck in a very small cycle of a very large search space graph.

Other results regarding use of the interface describe successful participants with more detail. They used graphical user interfaces much less; since Stata's user interfaces are simply visual representations of the same textual support systems (and often direct users to the textual systems once employed), these interfaces were largely a distraction. Interestingly, successful participants managed their screen real estate much more effectively. Since the Stata environment consists of a main window with a number of child windows, such management was crucial for being able to compare error messages to help files. Unsuccessful participants showed much less management, which made it more difficult for them to use the information available to them.

Finally, an interesting result was that throughout the problem, participants who successfully spoke much more. Although it is impossible to tell whether their thoughts are representative of their problem solving strategies, it does suggest that the successful

participants were thinking *more* about the problem, entertaining possible strategies to use, and interpreting feedback provided by Stata.

The final task in problem 1 of executing the t test, was largely dependent on the previous task of generating the random data. As shown in Table 7, there were only a scant number of participants who completed this final task without having completed the previous two tasks. Most likely, they struggled with creating the two sets of uniformly distributed data and tried to complete as much of the problem as possible.

Nevertheless, in general, one thing is apparent: the best predictors of success on the programming task was intelligence, irrespective of background (thought programming experience did help) Furthermore, participants who used problem-solving strategies that were appropriate to the environment and the task were the most successful. Since there were numerous users in the study who were not as intelligent and did not use appropriate strategies, an important problem is to help these unsuccessful participants to complete programming tasks similar to the one used in this study. Such a problem has numerous issues:

- Are there environments that force the use of environment- and task-appropriate strategies and if so, what are their characteristics? Is forcing the use of appropriate strategies effective?

- If the use of environment- and task-appropriate strategies cannot be enforced, are there environments that can teach the appropriate strategies either explicitly or implicitly? Are they feasible or distracting?

- How much does the presentation of information influence individuals trek through the information search space? If it has substantial influence, are there

effective ways to present it so that the search problem becomes as simple as it is for successful users, but without the prerequisite higher intelligence and knowledge of programming?

The Testing and Debugging Task

The testing and debugging task was inherently different in nature from the programming task. There was no logical sequence of tasks participants could follow, nor were their prescriptive guidelines of how to find information about what a "do-file" is or how it works. In effect, this task was an opportunity to see what participants would do with virtually no instruction and limited experience with Stata. Who were the participants that excelled at finding each specific bug? With respect to overall performance, the successful participants had higher self-reported math abilities and more positive attitudes towards computers and mathematics. Interestingly, there was only a weak relationship between programming experience and success. Before we make conclusions about why this is, let us consider each bug in turn.

Probably the least interesting bug inserted into the do-file was the for loop iteration bug: the for loop, which was to loop through outlier values from 1 to 10, was looping from 10 to instead ("for num 10/1: …" instead of "for num 1/10:…") . This had no consequence on the data produced, or on the graph produced. Thus it is technically a bug but only because it did not comply with the problem specification. However, this command was systematically constructed to be part of the longest and most complex command in the file, in order to see if complexity attracted attention. In fact, a third of the participants in the sample took the time to fix the problem despite its lack of real

problems. This result alone illustrates something inherent in testing and debugging: complicated segments of even a simple end-user program can become major time sinks.

Who were the individuals who successfully fixed this irrelevant bug? These participants were on the whole computer science and other majors—not psychology majors—suggesting that the programming experience either compelled them to inspect the command, or, once it was identified as not complying to the specification, to fix the insignificant problem. Across the whole problem, these tended to be individuals who spent much more time on the for loop command, used many more examples from the tutorial and the online help system, and sought more information. Furthermore, they spent less time inspecting the graph and the graph command, and more time on the *generate pvalues* and *generate valueofoutlier* commands, both of which had similar syntax similar to the range—and in fact, participants who fixed this problem either only fixed this bug, or also fixed one or both of these commands. This suggests that these individuals were fixated more on understanding the underlying syntax and language, and less on the task: fixing the graph that was produced.

The trends raised by this bug raises an important issue: if there are individuals who, because of their background experience, become obsessed with specific aspects of a task and thus do not complete their task, are there ways to direct their attention to more relevant aspects of the problem? Furthermore, considering that many of the participants who successfully fixed this bug were computer scientists, it illustrates the importance of not only supporting end users, but all individuals who have predispositions for distraction.

The next two bugs of interest were the *generate pvalues* and *generate valueofoutlier*

commands. Recall, both commands were nearly identical: the *generate pvalues* command

had the form "generate pvalues = 0 in 1/100" and the *generate valueofoutlier* command

had the form "generate valueofoutlier = _n in 1/100." The only differences between the

two commands were that the *generate valueofoutlier* command had the symbol "_n" in

it—which refers to the current observation, meaning the variable *valueofoutlier* would

contain the values from 1 to 100 in observations 1 through 100—and the command also

had a comment above it that said "* Generate a variable that contains 1 through 10;"

Each command was supposed to end in "1/10" in order to create the values 1 through 10

and store 10 p-values. Thus the bug in the *generate valueofoutlier* command should have

been more obvious because of the comment, but more complicated because none of the

participants knew what the symbol "_n" meant. Interestingly, despite the similarity of the

syntax of the two bugs, two thirds of the participants fixed the *generate valueofoutlier*

command while only a third fixed the *generate pvalue* command. Furthermore,

individuals who tended to fix the *generate valueofoutlier* bug tended to fix only this bug,

or either the loop iteration bug or the *generate pvalue* bug but not both. This result has a

number of interpretations. First, there may have been some participants who did not

understand why what they fixed were bugs, otherwise they would have fixed both.

Another interpretation is that the comment in the *generate valueofoutlier* bug made fixing

the bug possible without understanding of the syntax. A final interpretation follows from

the nature of the do-file: participants only had to fix one of the two bugs to get the graph

to display correctly, since Stata could not graph missing values. Thus the *valueofoutlier*

bug may have had a higher probability of being noticed.

Before we consider the implications of these results, who were the individuals who successfully fixed these bugs? Participants who fixed the *generate valueofoutlier* bug tended to have more math experience, and more positive attitudes towards math, but there were no other significant distinctions. Possibly, the extra mathematics experience made the connection between the comment and the command stronger, highlighting the discrepancy between the two. Differences in participants who fixed the *generate pvalue* bug are equally underdetermined: such participants, if psychology majors, tended to have less positive attitudes towards computers, and if male, had less programming experience. Such distinctions are hardly useful. What is interesting is that unlike performance on problem 1, there were no significant differences in intelligence, statistics software experience, programming experience, or computer experience. This seems to suggest that debugging the do-file required less skill, and something more in terms of problem solving skills not captured by the psychological tests used in this study.

This brings us to the problem solving aspects measured for problem 2. Participants who fixed the *generate valueofoutlier* bug tended to verbalize their thoughts more throughout the problem, spent more time inspecting the *generate pvalues* command, while using more commands to test syntax. They spent much less time on familiar commands and more time paying attention to the graph generated by the do-file, which possibly explains why they successfully found and fixed the bug. They exhibited more guess and check behavior—likely in trying new values for the observation range in the command—but performed less cyclic behavior. They also tended to use the problem specification early on but not later, when they spent more time changing ranges in commands. To generalize, participants who fixed the *generate valueofoutlier* bug seemed

to have some intuition that the problems lie in the *generate valueofoutlier* and *generate pvalue* commands, but did not understand quite what was wrong.

Did participants who fixed the *generate pvalue* command show the same trends? In fact, participants successfully fixing this bug tended to spend more time on familiar commands, more cyclic behavior, and more syntax confusion. However, they still spent more time attending to the graph produced by the do-file while spending less time throughout the problem guessing and checking. Thus participants who fixed this bug differ from participants who fixed the *generate valueofoutlier* bug in that they seemed to be more confused about the command. This could be because there was no comment to guide them to a possible solution. In fact, only half of the participants who solved the *pvalue* bug solved the *valueofoutlier* bug.

The final bug was the inverted axes on the graph produced by the do-file. The graph command in the do-file was at the bottom of the file and the graph, though it did have labels, looked much like the curve that participants were expecting. Thus for some reason, it seemed to be difficult to spot. Participants who were successful at fixing this problem had higher statistics competency and also tended to fix the observation range in the *generate valueofoutlier* bug, likely because the extra values created by the command became obvious once the graph was inverted. With respect to problem solving, these participants spent more time inspecting the graph command, ignoring the familiar commands, and more time inspecting the graph itself. Interestingly, they also tended to speak much more early in the problem as if considering their approach to debugging. This is confirmed by the fact that these participants spent more time using graphical user interfaces and searching for information early in the problem. As the most logical way to

approach the problem was to inspect the output, it is no wonder that few participants fixed the for loop iteration bug, which had no consequence on the graph.

Given the diversity of the bugs contained in the short do-file, the backgrounds of participants who successfully tested and debugged the do-file, and the complex combinations of bugs that participants solved, we can conclude a number of things about individual differences, testing, and debugging:

- The structure of the program and the relationships of the commands in the program largely determine the success of debugging
- Programming experience, along with intelligence and statistical competency played no role in predicting successful debugging
- Comments and large, complex commands attract attention, and this is sometimes useful and sometimes not
- Few individuals inspected the output of the program, but those who did tended to find the rest of the bugs

Given these observations, it is clear that testing and debugging are not as much of a science as programming is: no matter what the background and intelligence, there seem to be numerous strategies that participants employed, all with varying levels of success.

Are There Distinct Categories and Are They Useful?

The earlier discussion focused on using already established categories and measures to describe successful participants. In such exploratory research, it is also important to consider whether participants' interactions with Stata—independent of any categorizations or measures already established—cluster together into distinct categories

in their own right, and to determine if such categories have any relationship to these already established measures and categories.

Distinct Programming Strategies

As reported in the results, there were three types of programming strategies observed based on the data that was collected about participants' interaction with Stata on the programming task. Cluster p1 had the following properties:

- Less use of examples, problem specifications, and user interfaces
- Less cyclic behavior, attention to feedback, and guessing and checking
- Less trouble with syntax and less frustration

This group of participants, because they were successful, obviously knew how to solve the problem without support from the Stata environment and exhibited far more appropriate behaviors in the context of the task and environment. Cluster p1 also tended to have

- Far more successful on problem 1 as a whole, with most participants succeeding at each milestone in the problem
- Mostly computer science and other majors (not psychology)
- Higher intelligence and more programming, computer, and mathematics experience
- More positive attitudes towards computers and mathematics

An appropriate label for this group is *programmers*, given the strategies they used to solve problem 1 and their individual characteristics. Cluster p2 had the following properties:

- More use of examples, user interfaces, and specifications

- More cyclic behavior and attention to feedback; less guessing and checking

- Less trouble with syntax and slightly more frustration

This group seems to be composed of individuals who were either not very motivated or confused about how to begin solving the problem. They spent much more time with information that would not help time, repeated a lot of problem solving strategies despite the fact they did not help, but on the whole did not exhibit much problem solving. Cluster p2 tended to have

- Very limited success on problem 1 as a whole, with most only creating the data set of 1000 objects

- Mostly psychology and other majors (not computer science)

- Lower intelligence and less mathematics and programming experience

- More negative attitudes towards computers and mathematics.

An appropriate label for this group is *lost/unmotivated end users*. Cluster p3 had the following properties:

- Less use of examples and problem specifications

- More cyclic behavior, and much more guessing and checking

- Much more trouble with syntax, more information seeking

- Much more window management

This group seems to be composed of individuals who were motivated to solve the problem, search for information, and attempt many solutions; in other words, they exhibited unconstructive problem solving behavior. Cluster p3 tended to have

- Average success on problem 1, relative to the whole sample, with about a third succeeding on the whole problem

- Mostly psychology and other majors (not computer science)

- Equal intelligence to cluster p1, but less programming and mathematics experience

- More negative attitudes towards computers and programmers than cluster 1 but more positive than cluster p2

An appropriate label for this group is *lost/motivated end users*. In summary, there tended to be three clear types of participants in this study: users who knew what they were doing (programmers), users who were lost and unmotivated, and users who were lost but persistent. Clearly, as these three groups solved problem 1 using different methods, we must consider a number of questions:

- Does each group require different support systems in the end-user environment?

- If each group requires different support mechanisms, are there ways to detect these types of users reliably and non-obtrusively?

- If each group does not required different support mechanisms, what type of end-user environment can support all groups effectively?

Of course, data that would support all of these questions would describe these three groups with more detail in order to guide the approach to answering the questions. To some extent, the distinction of computer scientist and non-computer scientist helps predict problem-solving strategies, but there are clearly at least two types of end users.

Distinct Testing and Debugging Strategies

As reported in the results, there were three types of testing and debugging strategies observed based on the data that was collected about participants' interaction with Stata on the testing and debugging task. Cluster d1 had the following properties:

- More use of examples, and user interfaces, and less use of specifications

- More attention to familiar commands, and less attention to the *generate pvalues* command, the *generate valueofoutlier* command, and the for loop command

- Less attention to feedback from Stata and the graph produced

- Fewer changes to ranges in commands, fewer changes to comments, less guessing and checking

- Sought much more information

This group seemed to be composed of individuals who sought a lot of information, but payed little attention to the commands in the do-file that were unfamiliar and the graph produced by the do-file. Possibly these were individuals who lost sight of the original task—to find the bugs and fix the graph—and become more absorbed in understanding the language syntax. Cluster d1 tended to have

- An even mix of psychology, computer science, and other majors

- More failure than success at the bugs in problem 2, except on the *generate valueofoutlier* command.

- Average programming experience and slightly above average mathematics experience relative to the sample

An appropriate label for this group is *curious/distracted*, given the extent to which participants ignored the do-file and learned about the environment instead. Cluster d2 had the following properties:

- Less use of examples, user interfaces, but more use of specifications

- More attention to the *generate pvalues* command, average attention to the *generate valueofoutlier* command and less attention to the *graph command* and for loop command

- More attention to the graph produced

- Less use of commands (and thus less syntax confusion and less attention to Stata), more changes to comments, and slightly less changes to ranges and words

- Sought less information

This group seemed to be more focused on the problem than cluster d2 given thought they sought less information, referenced the problem specification more, and attended to the two most problematic bugs (the *pvalue* and *valueofoutilier* commands) as well as the graph. This group tended to have

- More psychology and other majors than computer scientists

- Individuals who fixed either the *generate pvalue* or *generate valueofoutlier* bug, and sometimes the inverted axes bug

- Much less programming experience, and slightly lower attitudes towards computers

An appropriate label for this group is *hesitant/focused* given that the group seemed to want to solve the problem, stayed focused, but was not very adventurous or creative. Cluster 3, as compared to clusters d1 and d2 had the following properties:

- Slightly more use of examples, less use of user interfaces, and less use of specifications

- Much more attention to the *generate pvalues*, *generate valueofoutlier*, for loop, and graph command bugs than familiar commands

- More attention to the graph produced and much more attention to feedback from Stata

- Much more changes of words and ranges in commands

- Much more use of commands (and thus more syntax confusion)

- More guess and check behavior, and much more cyclic behavior

- Sought less information

Cluster d3 seems largely different from clusters d1 and d2 in that every command was inspected and seemingly in more detail given that errors from Stata were noticed and the graph checked more. Cluster d3 participants also seemed to try many different possible solutions and use commands in Stata to test them. Cluster d3 tended to have

- Mostly computer science and other majors

- Individuals who fixed the *generate valueofoutlier* bug and sometimes the inverted axes bug

- Much higher mathematics and programming experience

- More positive attitudes towards the experiment

Given the enthusiasm and energy cluster d3 seemed to have, despite the group's lack of universal success at finding bugs in problem 2, an appropriate label for this group is *active/focused*.

With regard to testing and debugging strategies, there tended to be three clear types of participants: curious and distracted users, hesitant and focused users, and active and focused users. None of the groups were truly more successful than the other, except on the *generate valueofoutlier* command. Curiously, the curious and distracted users were more successful at spotting this bug. Furthermore, when it came to testing and debugging strategies, there were clear differences in programming experience but no clear differences in success. These results raise some important questions:

- Is there something about testing and debugging that makes it universally difficult, despite the testing and debugging strategy?

- Do the different groups require different testing and debugging support?

- Are there ways to detect these types of users reliability and non-obtrusively?

- Why did the computer scientists in the sample not more successful? Are the reasons the same for the rest of the sample?

Relationships Among Programming, Testing, and Debugging Strategies

The final topic of discussion regards whether or not there is a relationship between programming, testing, and debugging strategies. The results presented earlier in Table 25 showed that for problem 1 clusters,

- Cluster p1 (*programmers)* participants were equally distributed among problem 2 clusters (*curious/distracted*, *hesitant/focused*, *active/focused*)

- Cluster p2 (*lost/unmotivated)* participants tended to belong mostly to cluster d2 (*hesitant/focused)*

- Cluster p3 (*lost/motivated*) participants tended to belong mostly to cluster d2 (*hesitant/focused*)

There are a number of interesting implications from these results. The programmers of cluster p1 tended to have very uniform behavior in problem 1, but show much diversity when posed with a testing and debugging problem. This is possibly because computer science majors are rarely trained in testing and debugging at a university, but rather in industry, which explains the variation. It is also curious that the lost and unmotivated participants of problem 1 tended to become the hesitant and focused participants of problem 2. Possibly these individuals found the testing and debugging problem more approachable because there was no restriction on the sequence of events that had to be performed for success. For the same reason, the lost and *motivated* participants of problem 1 may have found problem 2 easier to approach. Finally, it is interesting that only the *programmers* of problem 1 were found in the *active/focused* group of problem 2. Possibly, the success the programmers found in problem 1 gave them more knowledge necessary to be engaged in problem 2; furthermore, the background characteristics of the *programmers* group would predict the type of behavior seen in the *active/focused* group.

Limitations

There are some obvious limitations to this study, given its exploratory nature. Seventy-five participants is a relatively small number to ensure the validity of the

clustering done to identify end-users' strategies, and also limited the detail with which each cluster could be described.

Furthermore, the domain of statistics and the programming environment of Stata are hardly representative of all domains and end-user programming languages. Some of the findings presented here may not hold in other domains and environments, particularly because the strategies found were completely dependent on measurements of participants' interactions with the Stata programming environment. For example, if a visual programming language had been used instead, there would have been no measures of the number of commands entered, which would have changed the findings dramatically. At the most, since participants' backgrounds in statistics did not seem to play a factor in participants' success or strategies, these results may generalize to other textual programming languages with similar levels of feedback and environmental support for programming, testing, and debugging.

The sample came completely from Oregon State University undergraduates, which threatens the external validity of the findings. Samples that showed greater variation in statistics knowledge may have revealed a significant influence on performance and strategy, suggesting that domain knowledge is important in determining strategies, but this could not be shown here. Another significant problem is that the majority of end-user programmers are not college students, but rather, individuals employed at businesses. Such individuals may have developed different strategies for dealing with computers, may have had more or less computer experience, and may have shown more anxiety towards computer use than the younger sample used in this study. Each of these may have affected the findings. Furthermore, the sample had a significant gender bias: there

were few female computer scientists and few male psychologists, which makes it difficult to conclude anything about gender differences. Finally, many of the participants also had very little experience with English, with suggests that their understanding of the tutorial, of the online help in Stata, and performance on the measures of intelligence may have all been biased.

There may have also been a significant source of noise from the coding and extraction phases. Vigilance on the part of the coders may have varied depending on their personal factors, which means that some actions may not have been captured accurately or at all. Furthermore, the limited tests for inter-rater reliability may threaten the validity of up to sixteen of the participant's data. In the data extraction process, the regular expressions used to define metrics may have under- or over-matched the actions in the transcripts, which could have significantly influenced the results. As stated before, each metric was run and the actions that the data extraction script was matching were inspected. Nevertheless, there was room for error.

The reliability of the statistics test used is also another significant problem. The results on the test showed that the sample had poor statistical knowledge in general, but poor reliability also suggests that the test is a poor measure of statistical knowledge. This limited our abilities to analyze the influences of statistical knowledge on performance and strategies for programming, testing, and debugging.

Finally, the two problems used to explore participant's strategies both seemed to have very low ceilings; in other words, end users all had difficulty completing the problems successfully, suggesting that either the statistical knowledge required to complete the problems was too high or the Stata programming environment was too difficult to use.

Even still, computer scientists in the sample had poor statistical knowledge yet still

succeeded, suggesting that statistics knowledge was not the limiting factor.

Implications, Future Directions, and Conclusions

This study intended to investigate two areas of end-user programming: the influence of individual differences on success and whether or not groups of programming, testing, and debugging style would naturally cluster together and provide predictive value. Eighty-six participants, from backgrounds of computer science, psychology, engineering and humanities completed at battery of psychological tests and attempted to complete a programming task and a testing and debugging task in Stata, a statistical programming environment intended for non-computer scientists.

There were a number of significant findings that suggest important directions in future end-user programming research. For example, the best predictor of successful programming was higher intelligence and the use of problem solving strategies that were appropriate to the environment and task. The significant finding, however, is that *not all users are of high intelligence, nor do they know what strategies are appropriate to the environment and task*. In fact, many users may not even have a strategy. We care far less about the successful *programmers* group and more about the users fell into the two largely unsuccessful groups in this study: *lost/unmotivated* and *lost/motivated end users*. With regards to end-user programming, there are some basic questions that still need to be answered about these individuals:

RQ1. Can these two groups be reliably detected?

RQ2. Are these distinctions robust across different end-user programming environments?

RQ3. Do these two groups require different environmental support than the

programmers group and than each other?

Although there are far more questions that can be considered from the results

presented in this paper, these seem to be the most compelling, given the wide range of

end-user programming environments that have attempted to alleviate the difficulty of

programming. If these groups can be detected and modeled, we may be able to determine

whether or not the groups need different types of environmental support and what types

in specific may help the different groups succeed. One promising method of modeling

these users problem-solving strategies would be to construct search spaces in the form of

directed (and certainly cyclic) graphs, much like those described in the discussion. Using

the data from this study, the complete search space for problem 1 could be illustrated and

the complete path for each participant described in detail. Constructing this type of data

would allow for deeper analyses of the rich data set gathered in this study.

Given the amount of work left to do to understand end-user programming, considering

the problem of end-user testing and debugging seems a bit hasty. Nevertheless, there are

some obvious next steps given the findings of this study. As noted before, the structure of

the program and the relationships of the commands in the program seemed to be the

gatekeepers to success, rather than the users themselves—this is supported by the fact

that programming experience and intelligence played no role in predicting successful

debugging of the do-file given to participants. Rather, we saw different factors affecting

success: the comment in the *generate valueofoutlier* command attracted much attention,

and deservedly so; however, the complexity of the for-loop attracted just as much, yet

careful inspection yielded no results. Furthermore, a seemingly obvious finding was that

those who tended to inspect the output of the program first found more bugs—yet few did. Even more compelling is that trained programmers, the computer scientists in the study, showed a wide array of testing and debugging strategies, and the same ones as psychology and other majors at that. Some were curious and distracted, trying to understand the commands and investigating some tangent of the problem forgetting their task. Others were more focused, but tended to make few attempts at solving the problem. Others were focused, but acted as if in sandbox, changing everything, and absorbing information. None of these groups were more successful than the other. Thus testing and debugging seems less of an end-user issue and more of a general problem in computer science. This leads us to some obvious first questions:

RQ4. Can these three groups be reliably detected?

RQ5. Are these groups robust across all end-user programming environments?

RQ6. Do these groups require different environmental support for testing and debugging, and to what degree?

Much like the method proposed for studying programming, directed graphs representing the search space for testing and debugging the file could be constructed and the paths that each participant took generated. This data would shed more light on the *sequences* of behavior that participants exhibited, rather than just *counts*. It would also shed light on the findings that the *lost/unmotivated* and *lost/motivated* participants tended to be only *hesitant/focused* participants for testing and debugging, highlighting the subtle differences between this group and the wide variety of strategies employed by the *programmers* of problem 1.

In summary, though the data collected in this study was limited to a single end-user programming environment, there are a number of questions that can be answered by deeper analysis. Hopefully, such analysis will answer some of the basic questions regarding end-user programming, testing, and debugging, so that leveraging theories and tools may be created.

Bibliography

Alavi, M. (1985). Some Thoughts on Quality Issues of End-User Development Systems. *Proceedings of the 21st ACM/SIGBDP Conference on Data Processing Personnel*, Minneapolis, Minnesota, May 2-4.

Anderson, J. R. (1984) Learning to Program in LISP. Cognitive Science, 8, 87-129.

Bonar, J., & Soloway, E. (1989). Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers. In E. Soloway & J. C. Spohrer (Eds.), Studying the Novice Progrmmer, pp. 325-353. Hillsdale, NJ: Lawrence Erlbaum Associates.

Brooks (1977) Towards a Theory of the Cognitive Processes in Computer Programming. International Journal of Man-Machine Studies, 9, 737-751.

Chang, S. (1990). Principles of Visual Programming Systems. Prentice-Hall, Englewood Cliffs, NJ.

Cheney, P. H., Mann, R. I. & Amoroso, D. L. (1986). Organizational Factors Affecting the Success of End-User Computing. Journal of Management Information Systems, 3(1), 65-80.

Cypher, A. (1993). Watch What I Do: Programming By Demonstration. Cambridge, MA: The MIT Press.

DeLone, W. H. (1988). Determinants of Success for Computer Usage in Small Businesses. MIS Quarterly, 12(1), 51-61.

Goodman, D. (1987). The Complete HyperCard Handbook. New York: Bantam Books.

Green, T. R. G. (1977). Conditional Program Statements and Their Comprehensibility to Professional Programmers. Journal of Occupational Psychology, 50, 93-109.

Green, T. R. G. (1983) Learning Big and Little Programming Languages. In: Classroom Computers and Cognitive Science (A. C. Wilkinson, ed.) Academic Press, New York, 71-93.

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework. Journal of Visual Languages and Computing, 7(2), 131-174.

Hoc, J.-M., & Nguyen-Xuan, A. (1990). Language Semantics, Mental Models and Analogy. In J.-M. Hoc, T. R. G.. Green, R. Samurçay, & D. J. Gilmore (Eds.), Psychology of Programming, pp. 139-156. London: Academic Press.

Howard, G. S. & Smith, R. (1986). Computer Anxiety in Management: Myth or Reality? Communications of the ACM, 29, 7, 611-615.

Igbaria, M. & Parasuraman, S. (1989). A Path Analytic Study of Individual Characterstics, Computer Anxiety and Attitudes Toward Microcomputers. Journal of Management, 15(30), 373-388.

Igbaria, M., Pavri, F. N. & Huff, S. L. (1989). Microcomputer applications: An Empirical Look at Usage. Information and Management, 16, 187-196.

Lewis, C., & Olson, G. M. (1987). Can Principles of Cognition Lower the Barriers to Programming? In G. M. Olson, S. Sheppard, & E. Soloway (Eds.), Empirical Studies of Programmers: Second Workshop, pp. 248-263. Norwood, NJ: Ablex.

Lieberman, H. (1997). The Debugging Scandal and What to Do About It, " Communications of the ACM. 1997. 40(4). 26-78. Special Issue.

Matta K. M. & Kern, G. M. (1991). Interactive Videodisc Instruction: The Influence of Personality on Learning. International Journal of Man-Machine Studies, 35, 541-552.

Myers, B. A. (1980). Displaying Data Structures for Interactive Debugging. XEROX Palo Alto Research Center, June.

Myers, B. A., Chandhok, R. and Sareen, A. (1988). Automatic Data Visualizationfor Novice Pascal Programmers. IEEE Workshop on Visual Languages, IEEE Computer Soceity, Pittsburgh, PA, Oct, 192-198.

Nardi, B. A. (1993). A Small Matter of Programming: Perspectives on End User Computing. Cambridge, MA: The MIT Press.

Newell, A., & Card, S. K. (1985). The Prospects for Psychological Science in Human-Computer Interaction. Human-Computer Interaction, 1(3), 209-242.

Pane, J. F., Ratanamahatana, C. A. and Myers, B. A. (2001). Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems. International Journal of Human-Computer Studies, 52(2), 237-264.

Panko, R. (1998). What We Know About Spreadsheet Errors. Journal of End-User Computing, Spring, 15-21.

Pea, R. (1986). Language-Independent Conceptual "Bugs" in Novice Programming. Journal of Educational Computing Research, 2(1).

Rojansky, S. (1997). Linux Apprentice: DDD—The Data Display Debugger. Linux Journal, 42es.

Rothermel, K. J. Cook. C. R., Burnett, M. M., Schonfeld, J. Green, T. R. G., and Rothermel, G. (2000). WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical

Evaluation. Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland, 230-239.

Sammet, J. E. (1981). The Early History of COBOL. In R. Wexelblat (Ed.), History of Programming Languages. New York: Academic Press.

Sinha, A., & Vessey, I. (1992). Cognitive Fit in Recursion and Iteration: An Empirical Study. IEEE Transaction on Software Engineering SE-18, 386-399.

Ungar, L., Lieberman, H. & Fry, C. (1997). Debugging and the Experience of Immediacy. Communications of the ACM, 40, 4, 39-43.

Vessey, I. & Galletta, D. (1992) Cognitive Fit: An Empirical Study of Information Acquisition. Information Systems Research 2, 63-84.

Zmud, R. (1979). Individual Differences and MIS Success: A Review of the Empirical Literature. Management Science, 25(10), 966-979.