RR-89-30

RESEARCH REPORT

DEVELOPING AND EVALUATING A MACHINE-SCORABLE, CONSTRAINED CONSTRUCTED-RESPONSE ITEM

Henry I. Braun
Randy Elliot Bennett
Douglas Frye
Elliot Soloway

Educational Testing Service
Princeton, New Jersey
June 1989

Developing and Evaluating a Machine-Scorable, ~~Constrained Constructed-Response Item~~

Henry I. Braun

Randy Elliot Bennett

Educational Testing Service

Douglas Frye

Yale University

and

Elliot Soloway

University of Michigan

Acknowledgements

Appreciation is expressed to Jim Spohrer of Yale University for his help in analyzing the faulty solutions data and his insights on programming knowledge and skill. Assistance in data analysis was provided by Minh Wei Wang and Bruce Kaplan. Hazel Klein and Terri Stirling were instrumental in organizing and managing the data collection effort. Thanks are due to Carl Haag of the AP program and to C. Victor Bunderson for their encouragement and support. Finally, we are indebted to the students and teachers of the Advanced Placement Program without whom this study would not have been possible.

Abstract

The use of constructed response items in large scale standardized testing has been hampered by the costs and difficulties associated with obtaining reliable scores. The advent of expert systems may signal the eventual removal of this impediment. This study investigated the accuracy with which expert systems could score a new, non-multiple choice item type. The item type presents a faulty solution to a computer programming problem and asks the student to correct the solution. This item type was administered to a sample of high school seniors enrolled in an Advanced Placement course in Computer Science who also took the Advanced Placement Computer Science (APCS) Test. Results indicated that the expert systems were able to produce scores for between 82% and 97% of the solutions encountered and to display high agreement with a human reader on which solutions were and were not correct. Diagnoses of the specific errors produced by students were less accurate. Correlations with scores on the objective and free-response sections of the APCS examination were moderate. Implications for additional research and for testing practice are offered.

Developing and Evaluating a Machine-Scorable, Constrained Constructed-Response Item

Constructed-response items offer the opportunity to present examinees tasks similar to those they encounter in education and work settings. This similarity enhances face validity--the perception among examinees, program sponsors, test users, and critics alike, that the test is measuring something important. In addition, constructed-response items may measure somewhat different skills than their multiple-choice counterparts (Ward, Frederiksen, & Carlson, 1980), offer a window onto the processes used to solve the problem (Birenbaum & Tatsuoka, 1987), and better predict some aspects of educational performance (Frederiksen & Ward, 1978). Finally, constructed-responses may reduce the susceptibility of some items to a popular multiple-choice test-taking strategy: working backwards from solution to question by substituting each response option in turn until the correct response is found. Given these potential benefits, there is good reason to explore the utility of constructed-response items for a variety of assessment purposes.

Though constructed-response items have compelling advantages, they have seen relatively limited use in large-scale testing programs. The primary difficulty has been the subjectivity and high cost associated with scoring; whether for national programs like the Scholastic Aptitude Test or for such locally-managed efforts as district-wide achievement testing, the costs associated with training human graders to

achieve acceptable levels of agreement and supporting them while they score thousands of exams are prohibitive.

With the advent of low-cost computing capability, and with advances in cognitive psychology and computer science, has come the expert system, a program designed to emulate in a very circumscribed domain, the actions of a human specialist (Waterman, 1986). With such systems, moderately complex ~~constructed-response items can be objectively and~~ automatically scored (e.g., Bennett, Gong, Kershaw, Rock, Soloway, & Macalalad, 1988), and there is good justification to believe that more complex ones will be scorable in the not-too-distant future.

An example of applying expert systems to the scoring of constructed-response items is found in PROUST and its progeny, MicroPROUST (Johnson, 1985; Johnson & Soloway, 1985). PROUST was developed to study the conceptual errors made by students in learning to program in Pascal. The program is comprised of 15,000 lines of LISP code and runs on a VAX minicomputer. MicroPROUST was developed as a portable demonstration of the concepts embodied in PROUST. It is one-tenth the size of its forebear and, as a consequence, less powerful in its analytical techniques.

PROUST and MicroPROUST attempt to find non-syntactic bugs in Pascal programs. Each system has knowledge to reason about selected programming problems within a framework called intention-based analysis (Johnson, 1985; Johnson & Soloway, 1985). Intention-based analysis is derived from research on

how experts comprehend programs (e.g., Soloway & Erlich, 1984). This research suggests that in debugging programs experts first attempt to map the program into a deep-structure, goal and plan representation. Goals are the objectives to be achieved in a program whereas plans are stereotypic means (i.e., a step-by-step procedure) for achieving those goals. Following the lead of experts, PROUST and MicroPROUST first attempt to identify the goals and plans that the student intended to realize in a program, and then to identify the bugs produced, where a bug is conceptualized as an unsuccessful or incorrectly realized plan for satisfying a goal.

To analyze a problem, PROUST or MicroPROUST first reads the problem specification contained in its knowledge base. This specification enables the system to know what goals the student should be attempting to achieve in writing a particular program. The system uses this goal specification, its plan and bug knowledge bases, and the student's code to construct the solution intended by the student. For example, part of the specification for a problem might include the goal, "to read in data." The system would use this goal to locate in its knowledge base a set of plans to achieve this result. Next, it would locate the code templates that instantiate each of these plans. Third, it would attempt to match a portion of the student's code to one of these code templates. If a match is found, the system can make inferences about the student's intentions with respect to this

code segment, for instance, what meaning to attribute to particular variables. On the basis of these inferences, the system can predict how these variables will be used in achieving the next goal needed to satisfy the problem specification. If these expectations are violated (that is, if an appropriate code segment cannot be found to match the templates associated with plans for achieving that next goal), an attempt is made to match the code segment against templates for buggy implementations of that plan. This goal-plan matching strategy provides considerable leverage; correct and incorrect plans can be put together in different combinations to handle the variety of responses generated by novice programmers.

MicroPROUST has been used in two projects involving constructed-response items. The first project was undertaken to test the applicability of expert systems to analysis of the free-response item type used in the College Board's Advanced Placement Computer Science (APCS) program and the technology's generalizability to similar item types in other content domains (Soloway, Macalalad, Spohrer, Sack, & Sebrechts, 1987). MicroPROUST was modified to score a demonstration set of student solutions to two APCS problems and to one problem in geometry; GIDE (Sebrechts, LaClaire, Schooler, & Soloway, 1986; Sebrechts, Schooler, & Soloway, 1987), an extension of MicroPROUST, was programmed to score demonstration solutions in algebra and statistics. In each case, the item presented the student with a task (e.g., a specification for a computer

program, an algebra word problem) and asked him or her to write a solution (e.g., a computer program, the set of equations needed to solve the algebra problem) which the appropriate expert system then would analyze. The system's analysis consisted of identifying and describing for the student any conceptual errors made in solving the problem.

The second study examined the extent of agreement between MicroPROUST and human readers in diagnostically and numerically scoring a range of solutions to each of the two APCS programming problems (Bennett, Gong, Kershaw, Rock, Soloway, & Macalalad, 1988). In this activity, MicroPROUST was able to analyze only 42% of the solutions it encountered in a cross-validation sample (it offered no analysis on the remaining papers). However, in those programs it was able to analyze, its performance was comparable in most respects to humans.

PROUST's effectiveness in diagnosing student's constructed responses has been evaluated using responses to a programming problem developed by Soloway and his colleagues (Johnson & Soloway, 1985). In this study, PROUST was able to produce a complete analysis for 79% of the programs given to it. For the remaining programs, it produced either a partial analysis (17%) or no analysis (4%). Because the problem used in this study is seemingly more complex than those used in the MicroPROUST studies, it is likely that PROUST's superior performance is due to its greater complexity and computing power. Even with these advantages, the proportion of papers

PROUST is able to analyze is probably not high enough to justify use in operational testing environments. MicroPROUST, which is the more portable and--because of its design--the more modifiable of the two, is even further from such performance levels.

It appears that the primary impediment to achieving higher success rates is that the task of writing a computer program is a relatively open-ended one that can be done correctly or incorrectly in a multitude of ways. It is plausible that a more constrained task--but one that retains the character of a constructed response--might afford expert systems a greater chance for successful analysis. One possible constrained constructed-response task is to present a completed, but incorrect, program and ask the student to correct it. Though a program is not actually written, this "faulty solution" task, in contrast to many multiple-choice formulations, calls upon skills central to effective programming. The purpose of this project was to evaluate the accuracy of expert systems in scoring the faulty-solution task and, secondarily, the meaning of scores from this task.

Method

Subjects

Subjects were located by sending letters of invitation to all Advanced Placement Computer Science (APCS) teachers who had 15 or more students enrolled in their classes or who had participated in the June 1987 reading of the APCS examination. This initial mailing was made to teachers at 112 high schools

throughout the United States. Teachers at 70 of these schools indicated an interest in having their classes participate. Data collection forms were mailed to these 70 schools with returns received from 59 schools for 916 students. Of these students, 737 were matched with APCS examination scores in ETS files and had complete data for the first of two faulty solutions; 734 of these also had complete data for the second faulty solutions problem.

Instruments

Constrained constructed-response items. In our earlier work (Bennett, Gong, Kershaw, Rock, Soloway, & Macalalad, 1988), students were asked to write a computer program in response to a specification (e.g., "write a program that rotates the elements of an array such that the element in the first position is moved to the second, the element in the second position in moved to third, ... and the element in the last position is moved to the first position"). To limit the range of answers but retain the advantages of constructed response, the task now was refined to require the student to correct a faulty program. Two tasks of this type were created, both adapted from existing problems. The first was an adaptation of the "Rotate" problem from the 1985 APCS examination. This problem was used in its free-response format in the study by Bennett et al. (described above), which provides a baseline for comparing the functioning of the expert system. The second problem, the "Rainfall" problem, was developed by Soloway and his colleagues and has been

studied extensively by them (Johnson, Soloway, Cutler, & Draper, 1983). Baseline data for the free-response version of this problem are provided by the Soloway and Johnson (1985) investigation previously described. The Rainfall problem tests more complex skills than the problems typically found on the APCS examination and should provide a better evaluation of the limits of the faulty solution format.

For each of these two problems, eight variants were developed in order to enhance the generalizability of the findings. Six of these variants contained a single bug and two variants contained three bugs each. All bugs were of a nonsyntactic nature; that is, the program was executable but produced a result that, at least under some circumstances, was different from that described in the problem specification.

Bugs were chosen to reflect three categories that have been found to capture most of the nonsyntactic errors produced by novices when writing programs (Spohrer, 1989). These categories were arrangement, completeness, and detail. An arrangement bug occurred when all of the parts of a program were present but not put together properly. A completeness bug existed when one component was missing. When a single part of a component was at fault (e.g., a variable, operator) and could be repaired by changing one word or operator, the bug fell into the last category.

Two bugs were selected from each category, for a total of six different bugs (one for each single-bug variant). Each of the triple bug variants contained one bug from each category.

One variant for each problem, along with the directions to the student, is presented in Appendix A.

Expert systems. Because each of the expert systems has associated with it specialized knowledge bases, PROUST was used for the Rainfall problem and MicroPROUST the Rotate problem. The knowledge bases for both systems were developed within the context of previous studies. They were constructed to provide the systems with enough understanding to analyze complete programs written in response to a given specification. The Rainfall knowledge base resulted from analysis of approximately 150 programs; the knowledge base for the Rotate problem was developed from 45 student papers. Neither knowledge base was expanded or modified in any way for the current study.

The analysis produced by MicroPROUST consisted of a diagnostic comment, which identified the presence of a specific fault in the student's solution, and a grade on a five-point scale for the 1-bug variants and on a six-point scale for the 3-bug variants. Differences in the scales emanated from the need to award points for correcting different numbers of seeded bugs and to deduct points for the expected introduction of different numbers of new bugs (e.g., students would be expected to introduce more new bugs in solving the 3-bug variants than in the 1-bug variants because of the added complexity of the former items). Both scales were set to range from 0-2, with a score of 2 indicating a

perfect solution. However, because of the aforementioned differences, scores from the two scales are not comparable.

Because of the manner in which PROUST was originally constructed, only diagnostic comments were generated by the program. To produce numerical scores, a sample of 292 student solutions to the Rainfall problem (143 1-bug and 149 3-bug) was rated on a five-point scale by one of the authors without reference to the diagnostic comments generated by PROUST. These human ratings were used in all analyses of the Rainfall problem that required a numerical score.

Advanced Placement Computer Science Examinations. Two Advanced Placement Computer Science Examinations are offered by the College Board: an "A" exam intended to assess mastery of topics covered in the first semester of an introductory undergraduate course in computer science, and an "AB" exam covering the full year's material. Computer Science "A" emphasizes programming methodology and procedural abstraction, but also includes the study of algorithms, data structures, and data abstraction. Computer Science "AB" includes all topics of Computer Science "A" as well as a more formal and in-depth study of algorithms, data structures and data abstraction. Computer Science "A" is comprised of 35 multiple-choice and 3 free-response items. Computer Science "AB" includes these items plus an additional 15 multiple-choice and 2 free-response questions. For this latter exam, both "A" and "AB" grades are reported.

Procedure

Each student was asked to respond to one variant of the Rotate problem and one variant of the Rainfall problem, where one problem contained three bugs and one contained a single bug. Problems were paired in counterbalanced order for a total of 24 combinations (2 problems x 6 single-bug variants x 2 triple-bug variants), with a single-bug variant always placed first. To give each problem set, or "packet," an equal chance of being administered, packets were mailed to schools in a "spiralled" fashion based on the number of APCS students at each site (e.g., combinations 1-18 mailed to school #1, 19-24 and 1-6 to school #2, and so on). Teachers were instructed to administer both problems in a single class period.

Each problem was presented on an 11" x 17" multi-layer form. The form was divided vertically into two halves, each of which had a triple-spaced copy of the faulty solution (see Appendix A). Students were given written instructions that presented the problem specification and directed them to modify the solution on the right half using the one on the left as a reference. Allowable modifications were limited to insertions and deletions.

When the student had completed the task, he or she was instructed to tear off the bottom layer of the sheet (which contained a copy of the original problem and a carbon of the corrections made by the student), and return the top half to the teacher for mailing to ETS. Correct answers were then to be given out by the teacher who was provided with a packet of

instructional suggestions for maximizing the use of the materials.

Data Analyses

Student responses were put into machine-readable format by transcribing the student's handwritten corrections. (The student's corrections were modified by the authors only where obvious, minor errors in program syntax were detected.) This ~~corrected program was analyzed by the appropriate expert~~ system, and in some cases hand-scored as described above. Two types of analyses were then conducted with each analysis run separately on the total group and on the "AB" group (i.e., those students taking the complete APCS examination). The first focused on the expert systems' success in analyzing student responses. For each system, the percentage of responses for which an analysis was produced was calculated. For both systems, these percentages are directly comparable to the systems' success in analyzing the free responses to the Rotate and Rainfall problems produced by earlier cohorts. These percentages were 42% for MicroPROUST in analyzing Rotate (Bennett, et. al, 1988) and 79% for PROUST's assessments of Rainfall (Soloway & Johnson, 1985).

The second analysis centered upon the meaning of scores from the faulty solutions item type. This analysis involved (1) estimating the agreement between human and machine ratings of students' responses to the item-type, and (2) computing the product-moment correlations between these scores and multiple-choice and free-response scores on the APCS examination.

To assess the rater reliability of scores assigned to the faulty solution problems, a sample of 84 responses to the Rotate problem was graded by one of the authors without knowledge of the scores assigned by MicroPROUST. The Pearson Product-Moment correlations between scores assigned by the human grader and the expert system were then computed.

Because PROUST does not generate numeric scores, a somewhat different approach to estimating rater reliability for the Rainfall problem had to be taken. First, 79 of the 292 responses that had already been handscored without reference to PROUST's comments were selected. The scores on these 79 papers served as human ratings. Next, a scoring component for PROUST was simulated by having one of the investigators read PROUST's comments--without knowing to which student's paper a set of comments referred--and assign a score to the paper based only on those comments. These two sets of scores were then correlated. This method is, at best, an approximation of the scores PROUST would assign if it had such capability and, hence, its results need to be carefully considered.

Once the correlations between human and machine scores were computed, the agreement levels for the Rotate and Rainfall problems were compared. This was accomplished by transforming the correlations to z-scores and testing this difference (McNemar, 1962).

Agreement was also assessed by tabulating the frequency with which a rater and the expert system concurred on whether

a paper was error free. For this analysis, a two-by-two contingency table was constructed and the proportion correct (i.e., the number of agreements divided by the number of agreements and disagreements), and Cohen's kappa were calculated. Kappa is the proportion of correct classifications beyond that expected by chance and can be tested statistically (Fleiss, 1981). In general, ~~statistically significant values greater than .75 may be taken~~ to represent excellent agreement, values between .40 and .75, fair to good agreement, and ones below .40 poor agreement beyond chance (Landis & Koch, 1977). Finally, the frequency with which the reader and system agreed on the diagnosis given individual bugs was tabulated. Both the contingency table analysis and the analysis of individual bugs were conducted on a sample of 186 solutions and were completed only for the Rotate problem and only for a combined sample of 1- and 3-bug variants.

The meaning of faulty solution scores was also assessed through correlational analyses. Using the Fisher *r*-to-*z* transformation, averages were computed for the correlations (1) among the free-response questions, (2) between Rotate and the free-response questions, (3) between Rainfall and the free-response questions, and (4) between the free-responses and the objective score. Selected averages were compared among themselves and with the individual correlations between each faulty solution and the APCS objective score.

Results

Tables 1 and 2 present APCS means and standard deviations for the two study samples and for the population taking the 1988 APCS examination. (Scores in this and all other analyses were originally derived from number-right raw score as opposed to the formula scores used in the APCS program.) For each score, sample means were tested for differences with the population mean which was treated as a population parameter. While several significant differences were observed, their magnitude was relatively small, ranging from 9% to 11% of a standard deviation on the "A" test, and from 10% to 14% of a standard deviation on the "AB" examination. The size of these differences suggests that the study sample did not dramatically differ in computer science knowledge from the population taking the test.

Insert Tables 1 and 2 about here

Table 3 presents data on the proportion of solutions that MicroPROUST and PROUST were able to analyze. Of the 737 students responding to the Rotate problem, MicroPROUST was able to provide an analysis for 614 or 83%. Of the 123 solutions it was not able to analyze, 18 were unparsable; that is, they were so poorly formulated syntactically, that the program rejected them outright. When the 105 parsable but ungraded programs were analyzed by a human grader (Spohrer, Frye, & Soloway, 1988), two findings emerged: (1) all

failures could be classified as due to incompleteness in MicroPROUST's knowledge base, and in the bulk of cases to a limited set of omissions, and (2) the overwhelming majority of solutions were wrong. With respect to the first point, 81 of the 105 analysis failures could be accounted for by 7 major classes of bugs. In fact, by adding a single new bug rule, MicroPROUST was able to analyze 30 more of the 105 solutions. On the second point, only 9 of the 105 programs were correct, organized in ways unknown to MicroPROUST. Adding in the unparsable solutions (which were by definition incorrect), 114 of the 123 analysis failures (93%) represented wrong solutions to the problem.

Insert Table 3 about here

PROUST was able to analyze 94% of the 734 Rainfall solutions it was given. PROUST's greater success rate was presumably due to its added flexibility and power. Because of its high success rate, an analysis of its failures was not conducted.

Aside from the overall difference between problems evaluated by PROUST and MicroPROUST, the rate of successful analyses held fairly constant across variants and study samples. The largest difference, between the Rotate 1- and 3-bug variants in the "AB" sample, was four percentage points.

Table 4 reports data on the agreement between scores assigned by humans and those assigned by the expert system.

For both the total sample and the "AB" sample, the agreement for the Rotate problem significantly exceeded that for Rainfall when all variants were combined within a problem (z = 3.56, $p < .001$ for the total sample; z = 3.68, $p < .001$ for the "AB" sample). When the variants were separated into 1- and 3-bug types, however, the correlations between the two 3-bug problems were no different (z = -.59, $p > .05$ for the total sample; z = -.42, $p > .05$ for the "AB" group), though the differences between Rotate and Rainfall remained for the 1-bug problem (z = 3.54, $p < .001$ for the total sample; z = 3.70, $p < .001$ for the "AB" sample). With the exception of the 1-bug Rainfall variant, the levels of agreement were comparable to those found for the Rotate problem in its fully free-response format (Bennett et al., 1988).

Insert Table 4 about here

Shown in Table 5 are the proportions of papers classified by MicroPROUST and by a reader as perfect or not (i.e., containing one or more bugs). For this sample, the observed proportion correct was .94 (the sum of the diagonal entries in table 5), indicating that in the overwhelming majority of cases the two raters agreed. Kappa for this table is .87 ($p < .001$, z = 4.85), suggestive of excellent agreement beyond chance.

Insert Table 5 about here

Although agreement on the dichotomous classification of papers was substantial, a lower level of agreement is evident when the individual bugs are considered. For this analysis, MicroPROUST and the reader agreed on the diagnosis of 384 bugs; that is, both gave the same location and interpretation. In 322 cases the reader and MicroPROUST disagreed: on 141 of these, the reader believed MicroPROUST's diagnosis of the bug to be spurious; the remaining 181 cases constituted bugs the reader believed to exist but MicroPROUST failed to confirm. Whereas such levels of disagreement may seem substantial, it is well to note that considerable disagreement in identifying individual bugs also appears among human readers (Bennett et al., 1988).

Table 6 presents the summary statistics for performance on the faulty solutions problems for the total student sample and for those taking the "AB" examination. Each problem is graded on a 0-2 scale (Rotate by MicroPROUST and Rainfall by a human rater). Because the two problems were graded by different mechanisms, and because the scales used for the 1- vs. 3-bug variants were different within problems, performance comparisons are best restricted to the same problem variant taken across samples. In these cases, the group taking the "AB" examination does marginally better than the total sample.

Insert Table 6 about here

The complete correlation matrices for the different item types are presented in Appendix B. Table 7 summarizes these matrices by showing selected mean and individual correlations between the faulty solution problems and the components of the APCS score, with the means computed using the Fisher r-to-z transformation. For example, the first entry in the first row, .46, is the mean of the correlations (.49, .43, .44 from Table 8, Appendix B) among free-response items #1, #2 and #3 for students in the total sample who took the 1-bug Rotate and the 3-bug Rainfall faulty solution items. The second entry in the first row, .50, has the same interpretation but is based on students in the total sample who took the 3-bug Rotate and the 1-bug Rainfall versions. Since these two groups are (approximately) random half samples, the two entries should be equal but for sampling fluctuations. The same is true of the pair of entries in the fourth row of the table.

Similarly, each pair of entries (row-wise) in the next two columns is based on random half samples of the "AB" group. The entries in the first row are the means of the correlations among the three free-response questions in the APCS "A" examination. Finally, each pair of entries in the last two columns is also based on random half samples of the "AB" group. However, the entries in the first row are now the

means of the correlations among the five free-response questions in the full APCS "AB" examination.

Comparing the second row to the first, we see that the mean correlation between scores on the Rotate problem and scores on the free-response items are just slightly lower than correlations among the free-response items themselves. On the other hand, correlations between scores on the Rainfall problem and the free-response items are substantially lower (see third row). There does not seem to be a simple explanation of this finding. While Rainfall was somewhat harder than Rotate, the standard deviations of the score distributions were similar. Moreover, only the 1-bug variants of the Rainfall problem had lower scoring reliability. It would be useful to collect data on other problems to better understand these relations.

Comparisons in the lower half of the table mirror those in the top half. Correlations between scores on the Rotate problem and the Objective score are somewhat lower than those between the free-response items and the Objective score. Correlations between scores on the Rainfall problem and the Objective score are substantially lower.

Two points concerning the Rotate problem are worth noting. First correlations with the Objective score are uniformly higher than the mean correlations with the free-response items. Second correlations involving the 1-bug variants are uniformly higher than those involving the 3-bug variant.

Insert Table 7 about here

Discussion

This study was motivated by a desire to develop a non-multiple choice item that could be reliably and accurately scored by computer. The availability of valid items of this type could potentially broaden the scope of standardized testing and open new vistas in the area of diagnostic assessment. Building on previous work on the scoring of Pascal programs, a new constrained free-response item type was developed and its amenability to automated scoring investigated. The item type required the student to debug a faulty program that was meant to accomplish a set series of tasks.

The results were quite encouraging. The percentage of student solutions that could be analyzed ranged from 82% to 97%. Most of the programs that could not be analyzed were incorrect. For those that could, the classification into correct or incorrect was highly accurate. The more fine-grained diagnosis of specific bugs was less accurate, but still quite promising. The cause and nature of this inaccuracy (i.e., the types and seriousness of the misdiagnoses) will need to be explored further.

These statistics represent a substantial improvement over the results reported for the scoring of unconstrained student solutions to similar problems. Moreover, neither PROUST nor MicroPROUST were modified for this experiment. It seems

likely that with some tuning and an expansion of the plan and bug catalogs, the success rate could be increased. Of course, interest centers not on these particular problems, or even variants employing different seeded bugs. Rather, we would want to demonstrate that these expert systems could be quickly "educated" to deal with entirely new problems, with comparable success rates. This goal represents one important direction ~~for future work.~~

An obvious limitation of a small-scale study such as this is that it raises many more questions than it can answer. Future studies will not only have to investigate the mechanics of gearing up to analyze many problems but also have to explore and corroborate the correlational patterns that were examined in Table 7. One obvious question is under what circumstances the single-bug or the multiple-bug formats are to be preferred. Do they have systematically different psychometric properties? Only added experimentation can provide answers.

Despite these limitations, much remains to be done with the data already collected. Before constrained free-response items can be incorporated into standardized testing programs, their construct validity must be further explored. The correlational analyses described above are only a first step. Additional steps include (1) a detailed substantive analysis of student solutions, with particular emphasis on comparing strategies on the free-response and constrained constructed-response items, and (2) the application of factor analytic

methods to investigate the psychometric relations among the three item types (multiple choice, free response, and constrained constructed-response).

On the basis of the evidence accumulated so far, it appears that the faulty solution item type represents a plausible complement to the standard item types now employed in the APCS. The work described above should further illuminate the differences and similarities among the item types.

The incorporation of the new item type into the APCS examination would have substantial effects. For the student it would give explicit recognition of the importance of the ability to debug programs. This, in turn, may affect the content of the APCS curriculum. For the APCS program, replacing some of the free-response items with machine-scored faulty solutions--which are relatively brief--might well facilitate the inclusion of more non-multiple choice questions in the exam. Moreover, the cost of scoring the exam would be decreased because of the reduced numbers of graders required.

Constrained constructed-response items, thought of more generally than simply as faulty solution problems, may play an important role in other settings. In computer-based systems in which assessment is linked to instruction, these items can serve a very useful function. For example, consider an expert system that presents students with a series of tasks in which each successive task depends on the responses to previous tasks. As soon as the tasks go beyond the conventional

multiple choice format, the system is faced with the burden of "understanding" the student's response before any inferences can be made.

If open-ended responses are permitted, the results may be effectively infinite in variety, presenting the system developer with a nearly impossible job. The introduction of constrained constructed-response items can substantially reduce that burden, as we have already seen. Further, analytic power might be achieved by controlling the presentation of different item formats. For example, students might be first routed from multiple choice to the constrained constructed-response format. Only when they perform at a sufficiently high level would they be permitted to tackle the free-response items.

The benefits of such a presentation strategy would be twofold. First, the students who reach the free-response items would be more likely to produce unconstrained solutions that could be analyzed by an expert system. Second, the system could, in theory, "learn" enough about the student's knowledge and style from the constrained format to improve its chances in interpreting the unconstrained solutions. While this scenario is entirely speculative, it does not appear to go much beyond present capabilities. Our task is to extend those capabilities to comfortably include these visions of future assessments.

References

Bennett, R. E., Gong, B., Kershaw, R. C., Rock, D. A., Soloway, E., & Macalalad, A. (In press). Assessment of an expert system's ability to automatically grade and diagnose students' constructed-responses to computer science problems. In R. O. Freedle (Ed), Artificial intelligence and the future of testing. Hillsdale, NJ: Lawrence Erlbaum Associates.

Birenbaum, M., & Tatsuoka, K. K. (1987). Open-ended versus multiple-choice response formats--It does make a difference for diagnostic purposes. Applied Psychological Measurement, 11, 385-395.

Fleiss, J. L. (1981). Statistical methods for rates and proportions. New York: Wiley.

Frederiksen, N., & Ward, W. C. (1978). Measures for the study of creativity in scientific problem solving. Applied Psychological Measurement, 2, 1-24.

Johnson, W. L. (1985). Intention-based diagnosis of errors in novice programs (Tech. Report No. 395). New Haven, CT: Yale University, Department of Computer Science.

Johnson, W. L., & Soloway, E. (1985). PROUST: An automatic debugger for Pascal programs. Byte, 10(4), 179-190.

Johnson, W. L., Soloway, E., Cutler, B., & Draper, S. (1983). Bug Collection I (Tech. Report No. 296). New Haven, CT: Yale University, Department of Computer Science.

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. Biometrics, 33, 159-174.

McNemar, Q. (1962). Psychological statistics. New York: Wiley.

Sebrechts, M. M., LaClaire, L., Schooler, L. J., & Soloway, E. (1986). Toward generalized intention-based diagnosis: GIDE. Proceedings of the 7th National Educational Computing Conference.

Sebrechts, M. M., Schooler, L. J., & Soloway, E. (1987, May). Diagnosing student errors in statistics: An empirical evaluation of GIDE (abstract). Proceedings of the Third International Conference on Artificial Intelligence and Education.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge (Research Report #16). New Haven, CN: Yale University, Department of Computer Science Cognition and Programming Project.

Soloway, E., Macalalad, A., Spohrer, J., Sack, W., & Sebrechts, M. M. (1987). Computer-based analysis of constructed-response items: A demonstration of the effectiveness of the intention-based diagnosis strategy across domains (Final Report). New Haven, CN: Yale University.

Spohrer, J. C. (1989). MARCEL: A generate-test-and-debug (GTD) impasse/repair model of student programmers (CSD/RR #687). New Haven, CN: Yale University, Department of Computer Science.

Spohrer, J. C., Frye, D., & Soloway, E. (1988). A note on one aspect of MicroPROUST's performance. Unpublished manuscript.

Ward, W. C., Frederiksen, N., & Carlson, S. B. (1980). Construct validity of free-response and machine-scorable forms of a test. Journal of Educational Measurement, 17, 11-29.

Waterman, D. A. (1986). A guide to expert systems. Reading, MA: Addison-Wesley.

Table 1

Means and Standard Deviations of the APCS "A" Examination for Study Samples and the APCS Population

	Group			
APCS Score	Total Test Population (N=10,719)	Total Student Sample (N=737)	"AB" Test Population (N=7,372)	"AB" Student Sample (N=617)
35-item Objective (scale = 0-35)				
Mean	16.1	16.8**	17.5	17.8
SD	6.5	6.5	6.3	6.3
3-item Free-response (scale = 0-27)				
Mean	11.0	11.5	12.6	12.8
SD	7.3	7.6	7.2	7.3
Composite (scale = 0-70)				
Mean	30.4	31.7*	33.9	34.5
SD	14.9	15.5	14.6	14.9
Free-response #1 (scale = 0-9)				
Mean	4.1	4.3	4.7	4.8
SD	3.5	3.7	3.5	3.6
Free-response #2 (scale = 0-9)				
Mean	5.3	5.6**	6.0	6.1
SD	2.9	2.9	2.7	2.7
Free-response #3 (scale = 0-9)				
Mean	1.6	1.7	2.0	1.9
SD	2.7	2.6	2.9	2.8

Note. All scores are calculated using number-right raw score.

*$p < .05$, two-tailed test of total student sample mean with total test population mean.

**$p < .01$, two-tailed test of total student sample mean with total test population mean.

Table 2

Means and Standard Deviations of the APCS "AB" Examination for Study Samples and the APCS Population

	Group	
APCS Score	"AB" Test Population (N=7,372)	"AB" Student Sample (N=617)
50-item Objective (scale = 0-50)		
Mean	26.2	27.1**
SD	8.8	8.6
5-item Free-response (scale = 0-45)		
Mean	16.2	17.0
SD	10.4	10.7
Composite (scale = 0-100)		
Mean	43.7	45.8**
SD	19.1	19.1
Free-response #1 (scale = 0-9)		
Mean	4.7	4.8
SD	3.5	3.6
Free-response #2 (scale = 0-9)		
Mean	6.0	6.1
SD	2.7	2.7
Free-response #3 (scale = 0-9)		
Mean	2.0	1.9
SD	2.9	2.8
Free-response #4 (scale = 0-9)		
Mean	2.0	2.4***
SD	2.8	2.9
Free-response #5 (scale = 0-9)		
Mean	1.5	1.8**
SD	2.4	2.4

Note. All scores are calculated using number-right raw score.

**p < .01, two-tailed test.
***p < .001, two-tailed test.

Table 3

Ability of PROUST and MicroPROUST to Analyze Student Responses to Faulty Solution Problems

Group	Total Number of Responses	Percent Analyzed	Percent Unanalyzed	
			Parsed	Unparsed
Total sample				
MicroPROUST				
Rotate (all)	737	83%	14%	2%
Rotate 1-bug	382	82%	15%	3%
Rotate 3-bug	355	85%	13%	2%
PROUST				
Rainfall (all)	734	94%	4%	2%
Rainfall 1-bug	353	95%	3%	1%
Rainfall 3-bug	381	93%	5%	2%
"AB" sample				
MicroPROUST				
Rotate (all)	617	85%	13%	2%
Rotate 1-bug	318	83%	15%	2%
Rotate 3-bug	299	87%	11%	2%
PROUST				
Rainfall (all)	614	95%	3%	2%
Rainfall 1-bug	297	97%	2%	1%
Rainfall 3-bug	317	94%	4%	2%

Note. Percentage totals may not sum to 100% due to rounding.

Table 4

Agreement Between Handscored and Computer Scored Student Responses to Faulty Solutions

Group	Product-Moment Correlation	N
Total sample		
Rotate (MicroPROUST)		
All variants	.86	84
1-bug	.88	40
3-bug	.82	44
Rainfall (PROUST)		
All variants	.62	79
1-bug	.51	42
3-bug	.86	37
"AB" sample		
Rotate (MicroPROUST)		
All variants	.87	70
1-bug	.90	32
3-bug	.83	38
Rainfall (PROUST)		
All variants	.60	68
1-bug	.49	37
3-bug	.86	31

Table 5

Proportions of Papers Classified by MicroPROUST and a Reader as Perfect or Imperfect (N=186)

	MicroPROUST		
Reader	Perfect Paper	Imperfect Paper	Total
Perfect Paper	.33	.01	.34
Imperfect Paper	.05	.61	.66
Total	.38	.62	

Table 6

Performance on Faulty Solution Problems

(Score Scale = 0 - 2.0)

Faulty Solution	Total Student Sample	"AB" Student Sample
Rotate (all variants)		
Mean	1.06	1.15
SD	.84	.82
N	614	524
Rotate 1-bug		
Mean	1.23	1.34
SD	.94	.91
N	314	265
Rotate 3-bug		
Mean	.89	.95
SD	.69	.67
N	300	259
Rainfall (all variants)		
Mean	.91	.98
SD	.75	.76
N	292	248
Rainfall 1-bug		
Mean	1.06	1.17
SD	.85	.83
N	143	122
Rainfall 3-bug		
Mean	.77	.79
SD	.61	.63
N	149	126

Table 7

Selected Mean and Individual Correlations for Faulty Solutions Problems and APCS Scores

	APCS "A"				APCS "AB" '	
	Total Sample		"AB" Sample		"AB" Sample '	
Correlation	1-bug Ro'te/ 3-bug Rain	3-bug Ro'te/ 1-bug Rain	1-bug Ro'te/ 3-bug Rain	3-bug Ro'te/ 1-bug Rain	1-bug Ro'te/ 3-bug Rain	3-bug Ro'te/ 1-bug Rain
Relations with Free Responses						
Mean Among Free Responses	.46	.50	.40	.47	.41	.44
Mean Between Rotate and Free Responses	.43	.40	.36	.34	.36	.33
Mean Between Rainfall and Free Responses	.22	.26	.22	.19	.23	.14
Relations with Objective Score						
Mean Between Free Responses and Objective Score	.61	.66	.58	.63	.57	.59
Between Rotate and Objective Score	.51	.47	.46	.39	.47	.37
Between Rainfall and Objective Score	.29	.35	.30	.25	.30	.28

Appendix A

Faulty Solutions Problems

Rotate Array Program

Program specification: A procedure is needed that rotates the elements of an array s with n elements so that when the rotation is completed, the old value of s[1] will be in s[2], the old value of s[2] will be in s[3],..., the old value of s[n - 1] will be in s[n], and the old value of s[n] will be in s[1]. The procedure should have s and n as parameters. It should declare the type Item and have s be of type List which should be declared as List = array[1..Max] of Item.

Instructions. On the next page is a PASCAL program that was written to conform to this specification. The program contains 1 to 3 bugs (errors). All of the bugs are located within the lines that are triple spaced. The bugs are not syntactic; the program will compile and execute, but it will not produce the desired results. On the program on the right, correct the bugs by deleting lines and/or inserting new ones. Use the program on the left as your reference copy (both programs are exactly the same). The insertions and deletions you make will be recorded on a carbon copy of the program that you may keep. To keep the copy legible, use scratch paper to work out the exact form of the code you wish to insert, and erase only when absolutely necessary.

To delete a line, place a **D** in the space before it and draw a line through the code like this:

D ~~s[i] := s[i - 1];~~

To insert a new line, write in the new code and then place an **I** in the space to the left of it. For example:

I s[i] := s[i + 1];

Do not use arrows to indicate where lines should be moved in the program; use the delete-and-insert technique instead. If you want to change part of a line, you should delete the whole line and insert the corrected one.

Remember to write your name, date of birth, and school at the top of each sheet and to print legibly.

YOU SHOULD TAKE NO LONGER THAN 20 MINUTES TO COMPLETE THIS PROBLEM.

Rotate Array Program

Please print the following information:

Last name: _____ First name: _____

Date of Birth (mm/dd/yy): _____ Name of school: _____

Reference Side
(Use this side for reference.)

```
 1  program foo (input, output);
 2      const
 3          Max = 100;
 4      type
 5          Item = integer;
 6          List = array[1..Max] of Item;
 7      var
 8          PassedAsS : List;
 9          PassedAsN : integer;
10      procedure Rotate (var s : list; n : integer);
11
12
13
14          var
15
16
17
18              temp : Item;
19
20
21
22              i : 1..Max;
23
24
25
26          begin
27
28
29
30              for i := n downto 2 do
31
32
33
34                  s[i] := s[i - 1];
35
36
37
38              s[1] := s[n]
39
40
41
42      end;
43  begin
44      Rotate(PassedAsS, PassedAsN)
45  end.
```

Answer Side
(Please mark your corrections on this side.)

```
 1  ___program foo (input, output);
 2  ___    const
 3  ___        Max = 100;
 4  ___    type
 5  ___        Item = integer;
 6  ___        List = array[1..Max] of Item;
 7  ___    var
 8  ___        PassedAsS : List;
 9  ___        PassedAsN : integer;
10  ___    procedure Rotate (var s : list; n : integer);
11  ___
12  ___
13  ___
14  ___        var
15  ___
16  ___
17  ___
18  ___            temp : Item;
19  ___
20  ___
21  ___
22  ___            i : 1..Max;
23  ___
24  ___
25  ___
26  ___        begin
27  ___
28  ___
29  ___
30  ___            for i := n downto 2 do
31  ___
32  ___
33  ___
34  ___                s[i] := s[i - 1];
35  ___
36  ___
37  ___
38  ___            s[1] := s[n]
39  ___
40  ___
41  ___
42  ___    end;
43  ___begin
44  ___    Rotate(PassedAsS, PassedAsN)
45  ___end.
```

Rainfall Program

Program Description. A weather station needs a program to keep track of daily rainfall. The program must allow the user to type in the rainfall every day. It should reject negative values, since negative rainfall is not possible. When the user types in '99999', a sentinel value, then the program should stop accepting input. At that time, the program should print out the number of valid days that were entered, the number of rainy days, the average rainfall per day over the period, and the maximum amount of rainfall that fell on any one day.

Instructions. On the next page is a PASCAL program that was written to conform to this specification. The program contains 1 to 3 bugs (errors). All of the bugs are located within the lines that are triple spaced. The bugs are not syntactic; the program will compile and execute, but it will not produce the desired results. On the program on the right, correct the bugs by deleting lines and/or inserting new ones. Use the program on the left as your reference copy (both programs are exactly the same). The insertions and deletions you make will be recorded on a carbon copy of the program that you may keep. To keep the copy legible, use scratch paper to work out the exact form of the code you wish to insert, and erase only when absolutely necessary.

To delete a line, place a **D** in the space before it and draw a line through the code like this:

D ~~While (DailyRainfall <> 99999) Do~~

To insert a new line, write in the new code and then place an **I** in the space to the left of it. For example:

I DailyRainfall := 0

Do not use arrows to indicate where lines should be moved in the program; use the delete-and-insert technique instead. If you want to change part of a line, you should delete the whole line and insert the corrected one.

Remember to write your name, date of birth, and school at the top of each sheet and to print legibly.

YOU SHOULD TAKE NO LONGER THAN 20 MINUTES TO COMPLETE THIS PROBLEM.

Rainfall Program

Please print the following information:

Last name: _____ First name: _____

Date of Birth (mm/dd/yy): _____ Name of school: _____

Reference Side
(Use this side for reference.)

```
 1  Program Rainfall(input,output);
 2    Var DailyRainfall,TotalRainfall,MaxRainfall,Average : Real;
 3        RainyDays,TotalDays : Integer;
 4    Begin
 5
 6
 7
 8      RainyDays:= 0; TotalDays:= 0; MaxRainfall:= 1;
 9
10
11
12      TotalRainfall:= 0; DailyRainfall := -1;
13
14
15
16      While (DailyRainfall <> 99999) Do
17
18
19
20        Begin
21
22
23
24          Writeln ('Please Enter Amount of Rainfall');
25
26
27
28          Readln(DailyRainfall);
29
30
31
32          If DailyRainfall >= 0 Then
33
34
35
36            Begin
37
38
39
40              If DailyRainfall > 0 Then RainyDays := RainyDays + 1;
41
42
43
44              TotalRainfall := TotalRainfall + DailyRainfall;
45
46
47
48              If DailyRainfall > MaxRainfall
49
50
51
52                Then  MaxRainfall:=DailyRainfall;
53
54
55
56              TotalDays := TotalDays + 1
57
58
59
60            End
61
62
63
64          Else
65
66
67
68            Writeln ('Rainfall Must Be Greater Than 0');
69
70
71
72          Average := TotalRainfall/TotalDays
73
74
75
76        End;
77
78
79
80      If TotalDays > 0 Then  Begin
81
82
83
84        Writeln('Average is: ', Average: 10:2);
85        Writeln('Maximum is: ', MaxRainfall: 10:2);
86        Writeln('Total Number of Days is: ', TotalDays);
87        Writeln('Total Number of Rainy Days is: ', RainyDays)
88      End
89    Else Writeln('No Valid Days Entered.');
90  End.
```

Answer Side
(Please mark your corrections on this side.)

```
 1  ____Program Rainfall(input,output);
 2  ____  Var DailyRainfall,TotalRainfall,MaxRainfall,Average : Real;
 3  ____      RainyDays,TotalDays : Integer;
 4  ____  Begin
 5  ____
 6  ____
 7  ____
 8  ____    RainyDays:= 0; TotalDays:= 0; MaxRainfall:= 1;
 9  ____
10  ____
11  ____
12  ____    TotalRainfall:= 0; DailyRainfall := -1;
13  ____
14  ____
15  ____
16  ____    While (DailyRainfall <> 99999) Do
17  ____
18  ____
19  ____
20  ____      Begin
21  ____
22  ____
23  ____
24  ____        Writeln ('Please Enter Amount of Rainfall');
25  ____
26  ____
27  ____
28  ____        Readln(DailyRainfall);
29  ____
30  ____
31  ____
32  ____        If DailyRainfall >= 0 Then
33  ____
34  ____
35  ____
36  ____          Begin
37  ____
38  ____
39  ____
40  ____            If DailyRainfall > 0 Then RainyDays := RainyDays + 1;
41  ____
42  ____
43  ____
44  ____            TotalRainfall := TotalRainfall + DailyRainfall;
45  ____
46  ____
47  ____
48  ____            If DailyRainfall > MaxRainfall
49  ____
50  ____
51  ____
52  ____              Then  MaxRainfall:=DailyRainfall;
53  ____
54  ____
55  ____
56  ____            TotalDays := TotalDays + 1
57  ____
58  ____
59  ____
60  ____          End
61  ____
62  ____
63  ____
64  ____        Else
65  ____
66  ____
67  ____
68  ____          Writeln ('Rainfall Must Be Greater Than 0');
69  ____
70  ____
71  ____
72  ____        Average := TotalRainfall/TotalDays
73  ____
74  ____
75  ____
76  ____      End;
77  ____
78  ____
79  ____
80  ____    If TotalDays > 0 Then  Begin
81  ____
82  ____
83  ____
84  ____      Writeln('Average is: ', Average: 10:2);
85  ____      Writeln('Maximum is: ', MaxRainfall: 10:2);
86  ____      Writeln('Total Number of Days is: ', TotalDays);
87  ____      Writeln('Total Number of Rainy Days is: ', RainyDays)
88  ____    End
89  ____  Else Writeln('No Valid Days Entered.');
90  ____End.
```

avwz

Appendix B

Correlation Matrices for APCS and Faulty Solution Problems

Table 8

Product-Moment Correlations Among APCS "A" and Faulty Solution Scores for Total Student Sample

Students Taking 1-Bug Rotate/3-Bug Rainfall Faulty Solution Variants						
Score	1	2	3	4	5	6
1. 35-item Objective	--					
2. Free-response #1	.56	--				
3. Free-response #2	.67	.49	--			
4. Free-response #3	.60	.43	.44	--		
5. Rotate	.51	.43	.48	.38	--	
6. Rainfall	.29	.15	.31	.20	.29	--

Students Taking 3-Bug Rotate/1-Bug Rainfall Faulty Solution Variants						
Score	1	2	3	4	5	6
1. 35-item Objective	--					
2. Free-response #1	.65	--				
3. Free-response #2	.69	.54	--			
4. Free-response #3	.64	.47	.50	--		
5. Rotate	.47	.43	.45	.32	--	
6. Rainfall	.35	.24	.35	.19	.26	--

Note. For upper half of table, N = 314 for all correlations except those with Rainfall for which N = 120. For lower half of table, N = 300 for all correlations except those with Rainfall for which N = 129. Students whose Rotate or Rainfall solutions could not be analyzed are excluded from the computation of all correlations.

Table 9

Product-Moment Correlations Among APCS "A" and Faulty Solution Scores for "AB" Student Sample

Students Taking 1-Bug Rotate/3-Bug Rainfall Faulty Solution Variants						
Score	1	2	3	4	5	6
1. 35-item Objective	--					
2. Free-response #1	.53	--				
3. Free-response #2	.63	.40	--			
4. Free-response #3	.58	.40	.41	--		
5. Rotate	.46	.34	.39	.36	--	
6. Rainfall	.30	.12	.34	.20	.32	--

Students Taking 3-Bug Rotate/1-Bug Rainfall Faulty Solution Variants						
Score	1	2	3	4	5	6
1. 35-item Objective	--					
2. Free-response #1	.60	--				
3. Free-response #2	.65	.50	--			
4. Free-response #3	.63	.44	.47	--		
5. Rotate	.39	.35	.37	.29	--	
6. Rainfall	.25	.14	.29	.13	.20	--

Note. For upper half of table, N = 265 for all correlations except those with Rainfall for which N = 104. For lower half of table, N = 259 for all correlations except those with Rainfall for which N = 112. Students whose Rotate or Rainfall solutions could not be analyzed are excluded from the computation of all correlations.

Table 10

Product Moment Correlations Among APCS "AB" and Faulty Solution Scores for "AB" Student Sample

Students Taking 1-Bug Rotate/3-Bug Rainfall Faulty Solution Variants								
Score	1	2	3	4	5	6	7	8
1. 50-item Objective	--							
2. Free-response #1	.52	--						
3. Free-response #2	.66	.40	--					
4. Free-response #3	.57	.40	.41	--				
5. Free-response #4	.52	.32	.36	.48	--			
6. Free-response #5	.57	.34	.38	.51	.45	--		
7. Rotate	.47	.34	.39	.36	.36	.34	--	
8. Rainfall	.30	.12	.34	.20	.17	.29	.32	--

Students Taking 3-Bug Rotate/1-Bug Rainfall Faulty Solution Variants								
Score	1	2	3	4	5	6	7	8
1. 50-item Objective	--							
2. Free-response #1	.61	--						
3. Free-response #2	.66	.50	--					
4. Free-response #3	.64	.44	.47	--				
5. Free-response #4	.48	.45	.40	.37	--			
6. Free-response #5	.55	.40	.39	.53	.43	--		
7. Rotate	.37	.35	.37	.29	.32	.34	--	
8. Rainfall	.28	.14	.29	.13	.10	.05	.20	--

Note. For upper half of table, N = 265 for all correlations except those with Rainfall for which N = 104. For lower half of table, N = 259 for all correlations except those with Rainfall for which N = 112. Students whose Rotate or Rainfall solutions could not be analyzed are excluded from the computation of all correlations.